UNIVERSITY OF VERONA

DEPARTMENT OF COMPUTER SCIENCE

GRADUATE SCHOOL OF NATURAL SCIENCES AND ENGINEERING

DOCTORAL PROGRAM IN COMPUTER SCIENCE

CYCLE XXXV

Fault-based Analysis of Industrial Cyber-Physical Systems

S.S.D. ING-INF/05

Coordinator: ______________________

Prof. Ferdinando Cicalese

Tutor: ______________________

Prof. Franco Fummi

Doctoral Student: ______________________

Dott. Nicola Dall'Ora

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License, Italy. To read a copy of the license, visit the web page:
http://creativecommons.org/licenses/by-nc-nd/3.0/

Attribution — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.

NonCommercial — You may not use the material for commercial purposes.

NoDerivatives — If you remix, transform, or build upon the material, you may not distribute the modified material.

© 2022

Fault-based Analysis of Industrial Cyber-Physical Systems — Nicola Dall'Ora
PhD Thesis
Verona, May 1, 2023
ISBN <ISBN>

Abstract

The fourth industrial revolution called *Industry 4.0* tries to bridge the gap between traditional Electronic Design Automation (EDA) technologies and the necessity of innovating in many industrial fields, e.g., automotive, avionic, and manufacturing. This complex digitalization process involves every industrial facility and comprises the transformation of methodologies, techniques, and tools to improve the efficiency of every industrial process. The enhancement of functional safety in Industry 4.0 applications needs to exploit the studies related to model-based and data-driven analyses of the deployed Industrial Cyber-Physical System (ICPS). Modeling an ICPS is possible at different abstraction levels, relying on the physical details included in the model and necessary to describe specific system behaviors. However, it is extremely complicated because an ICPS is composed of heterogeneous components related to different physical domains, e.g., digital, electrical, and mechanical. In addition, it is also necessary to consider not only nominal behaviors but even faulty behaviors to perform more specific analyses, e.g., predictive maintenance of specific assets. Nevertheless, these faulty data are usually not present or not available directly from the industrial machinery. To overcome these limitations, constructing a *virtual model* of an ICPS extended with different classes of faults enables the characterization of faulty behaviors of the system influenced by different faults. In literature, these topics are addressed with non-uniformly approaches and with the absence of standardized and automatic methodologies for describing and simulating faults in the different domains composing an ICPS.

This thesis attempts to overcome these state-of-the-art gaps by proposing novel methodologies, techniques, and tools to: model and simulate analog and multi-domain systems; abstract low-level models to higher-level behavioral models; and monitor industrial systems based on the Industrial Internet of Things (IIOT) paradigm. Specifically, the proposed contributions involve the extension of state-of-the-art fault injection practices to improve the ICPSs safety, the development of frameworks for safety operations automatization, and the definition of a monitoring framework for ICPSs. Overall, fault injection in analog and digital models is the state of the practice to ensure functional safety, as mentioned in the *ISO 26262 standard* specific for the automotive field. Starting from state-of-the-art defects defined for analog descriptions, new defects are proposed to enhance the *IEEE P2427 draft standard* for analog defect modeling and coverage. Moreover, different techniques to abstract a transistor-level model to a behavioral model are proposed to speed up the simulation of faulty circuits. Therefore, unlike the electrical domain, there is no extensive use of fault injection techniques in the mechanical one. Thus, extending the fault injection to the mechanical and thermal fields allows for supporting the definition and evaluation of more reliable safety mechanisms. Hence, a taxonomy of mechanical faults is derived from the electrical domain by exploiting the physical analogies. Furthermore, specific tools are built for automatically instrumenting different descriptions with multi-domain faults. The entire work is proposed as a basis for supporting the creation of increasingly resilient and secure ICPS that need to preserve functional safety in any operating context.

Abstract (Italian)

La quarta rivoluzione industriale, chiamata *Industria 4.0*, cerca di colmare il divario tra le tecnologie tradizionali di EDA e la necessità di innovare in molti settori, come ad esempio quello automobilistico, avionico e manifatturiero. Questo complesso processo di digitalizzazione coinvolge ogni impianto industriale e comprende la trasformazione di metodologie, tecniche e strumenti per migliorare l'efficienza di ogni impianto. Il miglioramento della sicurezza funzionale in tutte le applicazioni legate all'Industria 4.0 deve sfruttare gli studi legati alle analisi basate su modelli e dati legati ad ogni specifico ICPS. La modellazione di un ICPS è possibile a diversi livelli di astrazione, in base ai dettagli fisici inclusi nel modello e necessari per descrivere i comportamenti caratterizzanti del sistema. Tuttavia, costruire questi modelli è estremamente complicato perché un ICPS è composto da componenti eterogenei relativi a diversi domini fisici. Inoltre, è necessario considerare non solo i comportamenti nominali, ma anche i comportamenti difettosi per eseguire analisi più specifiche, ad esempio analisi legate alla manutenzione predittiva specifici asset. Generalmente però questi dati sui guasti non sono presenti o non sono resi disponibili direttamente dai macchinari industriali. Per superare queste limitazioni, la costruzione di un modello virtuale di un ICPS esteso con diverse classi di guasti consente di caratterizzare i comportamenti difettosi del sistema. In letteratura, questi argomenti sono affrontati con approcci non uniformi e con l'assenza di metodologie standardizzate e automatiche per descrivere e simulare i guasti nei diversi domini che compongono un ICPS.

Questa tesi cerca di superare queste lacune presenti nello stato dell'arte proponendo nuove metodologie, tecniche e strumenti per: modellare e simulare sistemi analogici e multidominio; astrarre modelli di basso livello a modelli comportamentali di livello superiore; astrarre modelli di basso livello a modelli comportamentali più astratti; monitorare sistemi industriali basati sul utilizzo di sensori industriali. In particolare, i contributi proposti riguardano la definizione di tecniche all'avanguardia per l'iniezione di guasti multidominio in un ICPS e la definizione di un framework per il monitoraggio di un ICPS. Partendo dallo stato dell'arte dei difetti proposti per descrizioni analogiche, si sono proposti dei nuovi difetti per rendere più completo lo standard IEEE P2427, attualmente in fase di revisione. Inoltre, vengono proposte diverse tecniche per astrarre un modello descritto a livello di transistor verso un modello comportamentale per accelerare la simulazione di modelli guasti. A differenza di quello che avviene nel dominio elettrico, in quello meccanico non si fa un ampio utilizzo di tecniche per iniezione di guasti. Pertanto, l'estensione dell'iniezione di guasti al dominio meccanico e termico permette di supportare la definizione e la valutazione di meccanismi per garantire la sicurezza funzionale di un intero ICPS. Una tassonomia di guasti meccanici è derivata sfruttando le analogie fisiche definite tra il dominio elettrico e quello meccanico. Inoltre, sono stati costruiti strumenti specifici per l'instrumentazione di modelli descritti con diversi linguaggi con guasti multidominio. L'intero lavoro viene proposto come base per supportare la creazione di ICPS sempre più resilienti e sicuri che devono preservare la sicurezza in qualsiasi contesto operativo.

Acknowledgements

In primo luogo, vorrei esprimere la mia sincera gratitudine al mio relatore, Prof. Franco Fummi per l'instancabile sostegno durante il mio dottorato e per aver creduto in me. La sua esperienza e spirito intraprendente mi hanno permesso di affrontare tematiche di ricerca totalmente nuove e di trovare le migliori soluzioni per portare innovazione allo stato dell'arte in questo ambito di ricerca.

In secondo luogo, vorrei esprimere la mia profonda gratitudine al Dott. Renaud Gillon per gli innumerevoli meeting settimanali che mi hanno permesso di acquisire nuove conoscenze e approfondire tematiche prettamente industriali legate al mondo dei semiconduttori. Vorrei ringraziare il Prof. Riccardo Muradore e Prof. Andrea Calanca del mio comitato interno di tesi, per i preziosi e costruttivi suggerimenti, commenti e incoraggiamenti che mi hanno permesso di concludere il mio percorso di dottorato. Inoltre, vorrei ringraziare il Prof. Nicola Mazzocca e Prof. Frank Oppenheimer per i puntuali commenti suggeriti per migliorare la comprensione della tesi, sperando che possa essere di aiuto per studenti e futuri ricercatori.

Devo ringraziare tutti i miei colleghi, per le lunghe giornate passate insieme a lavorare e a discutere riguardo le più disparate problematiche che ogni dottorando si trova ad affrontare lungo il proprio percorso di crescita umana e professionale.

Ultimo, ma non meno importante, vorrei ringraziare tutta la mia famiglia a partire dai miei genitori e mia sorella, per avermi sempre stimolato a ricercare la giusta strada. Vorrei inoltre ringraziare la mia fidanzata per l'incrollabile pazienza e supporto durante il mio percorso di dottorato che mi ha aiutato a pensare sempre in modo positivo e costruttivo, giorno dopo giorno, senza mai perdere di vista il traguardo finale.

Grazie a tutti voi,

Nicola

Contents

List of Figures . . . VI

List of Tables . . . VII

List of Listings . . . X

List of Acronyms . . . XI

1 Introduction . . . 1
- 1.1 Overview . . . 1
- 1.2 Objectives . . . 2
- 1.3 Thesis Outline . . . 3

2 Background . . . 5
- 2.1 Industrial Cyber-Physical System (ICPS) . . . 5
- 2.2 Functional safety . . . 6
- 2.3 Digital twin . . . 7
  - 2.3.1 Maintenance process . . . 9
- 2.4 Fault modeling, injection, and simulation . . . 10

3 Analog fault modeling . . . 13
- 3.1 Analog defect modeling: investigation on realistic stuck-on/off defects . . . 14
  - 3.1.1 State of the art and definitions . . . 15
  - 3.1.2 Proposed stuck-on/off defect models . . . 16
  - 3.1.3 Experimental validation . . . 18
- 3.2 Analog defect injection and fault simulation techniques . . . 21
  - 3.2.1 State of the art techniques in defect injection and fault simulation . . . 22
  - 3.2.2 Research/tooling gaps and future directions . . . 27
- 3.3 Automatic tool for manipulating transistor-level netlist . . . 29
  - 3.3.1 State of the art and definitions . . . 30
  - 3.3.2 Language independent framework . . . 32
  - 3.3.3 Framework validation . . . 37
  - 3.3.4 Framework applications . . . 39
- 3.4 Verilog-A implementation of generic defect templates for analog fault injection . . . 43
  - 3.4.1 State of the art and definitions . . . 44

3.4.2 Verilog-A based fault templates . . . 47
3.4.3 Manipulation framework . . . 48
3.4.4 Experimental validation . . . 49
3.5 Analog fault grouping . . . 51
3.5.1 State of the art and definitions . . . 52
3.5.2 Impact of a fault on the AC matrices . . . 55
3.5.3 Predictive fault grouping . . . 56
3.5.4 Methodology validation . . . 58

4 Multi-domain fault modeling . . . 63
4.1 State of the art and definitions . . . 64
4.1.1 Multi-domain physical modeling . . . 64
4.1.2 Mechanical Faults . . . 66
4.1.3 Electrical Faults . . . 68
4.1.4 Behavioral fault modeling . . . 68
4.1.5 Fault Injection . . . 69
4.1.6 Failure Mode and Effect Analysis (FMEA) . . . 70
4.1.7 HIFSuite framework . . . 71
4.2 Represent a mechanical model with its equivalent electrical model . . . 72
4.2.1 Mechanical Networks . . . 73
4.2.2 Impedance Analogy . . . 73
4.2.3 Mobility Analogy . . . 78
4.3 Inferring mechanical fault models from the electrical domain . . . 79
4.3.1 Fault Modeling in Verilog-AMS . . . 80
4.3.2 Exemplification on the double-RLC circuit . . . 81
4.3.3 A new mechanical fault taxonomy . . . 85
4.4 Multi-domain fault modeling . . . 86
4.4.1 DC Motor . . . 87
4.4.2 DC motor with gear train . . . 87
4.4.3 Multi-domain fault injection and simulation . . . 89
4.5 Automatic tool for fault injection and simulation . . . 93
4.5.1 Fault injection tool implemented inside HIFSuite framework . . . 96
4.5.2 Testbench qualification . . . 99
4.6 Inferring thermal fault models from the electrical domain . . . 99
4.6.1 Physical analogies . . . 101
4.6.2 Realizing thermal models through analogies . . . 102
4.6.3 Thermal fault classification . . . 106
4.6.4 Experimental validation . . . 107

5 Model abstraction . . . 111
5.1 Abstraction of PWL models to C++ . . . 111
5.1.1 State of the art and definitions . . . 112
5.1.2 Abstraction methodology . . . 113
5.1.3 Methodology validation . . . 115
5.1.4 Methodology validation . . . 115

5.1.5 Integration into a Virtual Platform (VP) . . . 119
5.2 Abstraction of non-linear models . . . 119
5.2.1 Behavioral modeling . . . 119
5.2.2 Abstraction flow . . . 121
5.2.3 Verilog-A module example . . . 124

6 Application to Industrial Cyber-Physical Systems . . . 127
6.1 Modeling Industrial Cyber-Physical Systems . . . 128
6.1.1 Systems modeling for digital twin construction . . . 129
6.1.2 System-level simulation . . . 130
6.1.3 Faults modeling . . . 131
6.1.4 Coupling digital twins with faults . . . 133
6.2 Smart monitoring of Industrial Cyber-Physical Systems . . . 135
6.2.1 State of the art and definitions . . . 137
6.2.2 Data fusion in Industry 4.0 . . . 138
6.2.3 Extract knowledge with ML . . . 141
6.3 Smart monitoring of a production line . . . 146
6.3.1 Data collection and management . . . 147
6.3.2 Energy monitoring . . . 148
6.3.3 Case study: Industrial Computer Engineering Laboratory (ICELab) . . . 149
6.3.4 Online monitoring and data collection . . . 150
6.4 The design of a digital twin for predictive maintenance . . . 158
6.4.1 State of the art and definitions . . . 159
6.4.2 From sensor data to severity levels . . . 161
6.4.3 Monitoring State Machine (MSM) . . . 162
6.4.4 Predictive Maintenance Supervisor (PMS) . . . 164
6.4.5 Alternative use of MSM and PSM . . . 166
6.4.6 Methodology application . . . 167

7 Conclusions and suggestions for future research . . . 171
7.1 Summary of the proposed approaches . . . 171
7.2 Directions for future research . . . 172

Summary of the proposed innovative contributions . . . 173

References . . . 177

Appendices . . . 193

A Verilog-A fault templates . . . 193

B Verilog-AMS faulty circuits . . . 197

List of Figures

1.1 Overview of the proposed methodology, where two flows meet in the middle to produce different outcomes: ICPS analysis, functional safety, predictive maintenance, production optimization. 3

2.1 Evolution of the *Industry* from 2000 (Industry 3.0) to nowadays (Industry 5.0). 5
2.2 Functional safety standards subdivided for the different industrial sectors. 6
2.3 Physical dimensions that can be considered for the digital twin construction: functional behavior, energy consumption, and network communication. 8

3.1 Transistor-level description of the OPAMP model extracted from the analog benchmarks suite. 14
3.2 Transistor-level description of the comparator model extracted from the analog benchmarks suite. 14
3.3 Overview of the entire workflow. 15
3.4 Locations of possible analog defects of a transistor-level description of an inverter. 16
3.5 MOS cross-section with oxide trapped charges and interface traps. 16
3.6 Proposed MOS stuck-on defect model. 17
3.7 Proposed MOS stuck-off defect model. 17
3.8 PMOS ID(VG) characteristics for fault-free, hard-on/off, and stuck-on/off. 18
3.9 NMOS ID(VG) characteristics for fault-free, hard-on/off, and stuck-on/off. 18
3.10 Fault equivalence plot relative to the P2427 defects and the proposed stuck-on/off defect models for the Operational Amplifier (OpAmp) circuit. 19
3.11 Fault equivalence results between P2427 defects and the proposed stuck-on/off defect models for the Comparator (COMP) circuit. 19
3.12 Simulation performance/accuracy of different modeling techniques. 20
3.13 Fault simulation techniques. 22
3.14 Representation of multi-level modeling. 23
3.15 Overview of different fault grouping techniques. 24
3.16 Fault sensitivity analysis overview (see [1]). The left side shows the calculation of the results at every time point, while the right side shows the calculation of results only on selected test points. 27

3.17 Comparison between the different fault simulation techniques using different criteria: the ability to reduce the number of defects, fault injection throughput, ability to generate abstract defect models, structural simplification, convergence stability, reuse of partial results, and support Analog and Mixed-Signal (AMS) simulation. The score is related to different criteria used to evaluate the different simulation techniques, and it goes from zero to five, where zero means that the corresponding technique does not meet the required capabilities, while five means that the technique can efficiently implement the required capabilities. 28
3.18 Overview of the proposed manipulation framework. 30
3.19 Unified Modeling Language (UML) Class diagram of the internal structure of the framework - part 1. 35
3.20 UML Class diagram of the internal structure of the framework - part 2. 36
3.21 Abstract syntax tree of the OPAMP `xd1` component described in Eldo language. 38
3.22 Abstract syntax tree of the OPAMP `xd1` component described in Spectre language. 39
3.23 Comparison of the VOUT behavior for the transient simulation of the OPAMP described both in Eldo (identified by a blue line with crosses) and Spectre (identified by a purple line with circles). 40
3.24 Proposed MOS stuck-on defect model. 40
3.25 Proposed MOS stuck-off defect model. 40
3.26 Graphical visualization of the netlist presented in Listing 3.9 produced through the proposed framework. 42
3.27 Methodology for describing generic defect templates and injecting them into a transistor-level netlist. 43
3.28 Metal-Oxide-Semiconductor (MOS) type -n and type -p. 44
3.29 Some examples of faults in a N-Channel Metal-Oxide-Semiconductor Field-Effect Transistor (MOSFET). 45
3.30 OPAMP output waveforms for the different faults injected on the `mpd11` MOS component. . . 51
3.31 OPAMP output waveforms for the different faults injected on the `mpd12` MOS component. . . 51
3.32 Overview of the proposed methodology to group equivalent fault behaviors. 52
3.33 A 3-terminal circuit for S-parameter extraction. 53
3.34 Our implementation of an S-parameter port connected only with two terminals inside an analog circuit. 54
3.35 AC trajectories of the fault D20 (MOS drain-source short on the transistor `MNM12`) at three frequencies and four operating points. 58
3.36 Circles generated for the faults #D125, #D152, #D20 and #D75 relatives to the OpAmp circuit. These circles are generated with the circle-fitting algorithm for two operating points (columns) and two frequency points (rows). 59
3.37 Comparison between AC and transient-based grouping (with 15 clusters). 59
3.38 Comparison between the circle and transient-based grouping (with 15 clusters). 60

4.1 Overview of the proposed multi-domain fault injection into a Cyber-Physical System (CPS). 63
4.2 Fault locations in a simple electrical circuit. 69
4.3 HIFSuite overview 71
4.4 Analogies between mechanical (left) and electrical (right) system. 72
4.5 Mass-Spring-Damper mechanical network. 73

4.6 Mass-Spring-Damper schema. 74
4.7 The RLC circuit, the electrical equivalent of the Mass-Spring-Damper system according to impedance analogy. 75
4.8 Tuned Mass-Spring-Damper schema. 76
4.9 Tuned Mass-Spring-Damper mechanical network. 76
4.10 Electrical equivalent of the tuned Mass-Spring-Damper system (double-RLC). 76
4.11 The parallel RLC circuit, electrical equivalent of the Mass-Spring-Damper system according to mobility analogy. 78
4.12 Flow of proposed methodology to derive mechanical fault modes. It starts from a mechanical system, which is transformed into an electrical equivalent one, then faults compliant with the *IEEE P2427* standard are injected and simulated. The results are used to build a taxonomy of mechanical faults implied by the equivalent electrical model. 79
4.13 Normal response of the Double-RLC fault-free. 82
4.14 Response of the faulty Double-RLC, open. 83
4.15 Response of the faulty Double-RLC, short. 83
4.16 Response of the faulty Double-RLC, voltage source. 84
4.17 Response of the faulty Double-RLC, current source. 84
4.18 Fault analogy described by the mechanical taxonomy. 86
4.19 DC motor represented as an equivalent circuit of the armature windings and the free-body diagram of the rotor. 87
4.20 DC motor with a gear train, the dc motor (on the right top of the figure) is connected with the gear train (center of the figure). 89
4.21 Angular velocity of the motor, with a ramp as input. 91
4.22 Angular velocity of the gear train, with a ramp as input. 92
4.23 Angular velocity of the motor, with a sinusoidal input. 92
4.24 Angular velocity of the gear train, with a sinusoidal input. 92
4.25 DC motor plus wheel: the left part show the electrical circuit used to control the motor with a potential source, while the left part shows the shaft connected to a wheel. The EMF convert the energy from the electrical to the mechanical domain. 93
4.26 Full electric description of the DC motor plus one wheel. 94
4.27 Framework flow. 94
4.28 Simulation flow. 94
4.29 HIFSuite architecture including the new fault injection tool. 97
4.30 Overview of the modules that compose a general testbench. 99
4.31 Fault injection process in a multi-domain system, followed by the simulation of the fault scenarios. 100
4.32 Example of a Cauer network. 102
4.33 Example of a Foster network. 102
4.34 A DC Motor, an example of a multi-domain system (electrical, mechanical, thermal). 104
4.35 Cauer network as the thermal model of the DC-Motor. 104
4.36 Excerpt of the evolution of the DC-Motor power loss (upper graph) and internal mechanical parameters Ra (armature resistance) and Kt (torque constant) affected by an external heat source fault. 108
4.37 Angular velocity of the DC-Motor in fault-free and faulty scenarios. 109
4.38 Temperature of the DC-Motor's rotor in fault-free and faulty scenarios. 110

4.39 Temperature of the DC-Motor's shaft in fault-free and faulty scenarios. 110

5.1 Overview of the entire workflow. 112
5.2 Proposed methodology flow to abstract Piecewise Linear (PWL) descriptions. 113
5.3 Process flow used to abstract the model equations from Verilog-AMS to C++. 113
5.4 Simulation results for the half-wave rectifier, the memristor, and the ideal relay. 117
5.5 Circuit simulation setup. 118
5.6 Heterogeneous virtual platform. 119
5.7 Overview of abstraction of non-linear models to behavioral model. 120
5.8 Abstraction flow of non-linear models to behavioral model. 121
5.9 Schematic of a DUT with two ports with probes. 123
5.10 Interpolation between the local responses. 124
5.11 Schematic of a simple Wiener-type NL system. 125

6.1 Challenges on building and simulating a digital twin for a production line (composition of ICPS) that integrate faults. What are the best solutions? 128
6.2 Different strategies to combine digital twins and multi-domain fault models to assess the functional safety of an entire production line. 129
6.3 Multi-domain fault injection into a CPS (e.g., an anthropomorphic manipulator with its digital control). 133
6.4 Direct Current (DC) motor plus inertia modeled in Simulink/Simscape. 134
6.5 Overview of the proposed data fusion infrastructure that fuses sensor data with recipe data extracted from the Manufacturing Execution System (MES) inside the Data Integration HUB (DIH) module, avoiding time-consuming and error-prone manual annotation of the dataset, allowing enhanced extraction of knowledge with Machine Learning (ML). 136
6.6 Role of the DIH: it takes in inputs data coming from sensors (potentially at different frequencies, here gyroscopes, accelerometers, and power sensors, left bottom) and commands originating from the Automation Controller (with relative parameters, left top) and fuses all the data traces by automatically annotate them with production information. . . . 139
6.7 Automation pyramid: subdivision of each architectural level inside a production plant. The Automation Controller is positioned at the same level as the SCADA/HMI, and work as a supervisor that automatizes the production. 140
6.8 Representative examples of the raw signals provided by the sensors. The horizontal bar indicates the univocal *action-parameters* tuple of the Kuka at a given time frame. Note that the set of the univocal tuples *action-parameter* establishes the desired categorization for the 31 machine's states. 142
6.9 Overview of the ML pipeline. The raw signal (e.g., *sensor_id5_AngY*) is windowed based on the robot's actions and associated with corresponding labels. Then, it is fed into the TSFEL library to extract statistical features, which are reduced to a 50-dimensional set by feature selection. The final feature set is fed into a classifier. 143
6.10 Mean cross-validation accuracy of different machine state classifiers at different sampling rates of the sensed data. Error bars indicate the standard deviation of accuracy among different classes. 144
6.11 Sensors/physical properties color-mapped by their relevance. For sensor positioning, refer to the top-left section of Figure 6.6. 144

6.12 Distribution of the confidence values (Maximum Softmax Response (MSR)) of Bayesian Multilayer Perceptron (MLP-B) predictions for the normal events (left) and for the anomalies (right). The threshold C_{TH} is indicated by an arrow. 145
6.13 ICELab digital twin. 146
6.14 Structure of the ICELab production line. 150
6.15 ICELab production line schematical view. Each block composing the figure describes an industrial machinery available in the laboratory. All the machineries composing ICELab are directly connected with the conveyor belt that transports mini-pallets. 151
6.16 Data collection architecture of the ICE laboratory. Each industrial equipment composing the laboratory has on board an OPC UA server that can be implemented on the computer controlling the machine or into the Programmable Logic Controller (PLC) that support this protocol. The data collection by exploiting an equal number of OPC UA clients with respect to the machinery allows for retrieving all the interesting information from the machinery. This information is processed into specific Pub/Sub brokers enabling to process of large quantities of data. All the data are then saved into a time-series database, enabling to record of all the information with its specific timestamp associated. 152
6.17 Power model for the quality check station: active power (blue line), estimated power consumption $\hat{y}[t]$ (Equation 6.1), and allowed prediction interval $\hat{y}[t] \pm 3 \cdot \sigma$ (light blue area). 154
6.18 Kuka measured (solid blue) and estimated (dashed red) power consumption (top) and error of the prediction model (bottom). 155
6.19 ABB measured (solid blue) and estimated (dashed red) power consumption (top) and error of the prediction model (bottom). 156
6.20 Organization of the transport line. The numbers identify the motors monitored in this analysis: the main belt motors ($M1$ and $M2$) and the motors going to the robot assembly bay ($M7$, $M8$, and $M12$). 157
6.21 Transport line Power Consumption (blue line) and application of the prediction model: the red line is the estimated power consumption $\hat{y}[t]$, while the light blue area represents the allowed prediction interval. Motors $M1$ and $M2$ are always active; green labels indicate the activation of additional motors. 157
6.22 Example of adoption of the digital twin of power consumption for anomaly detection: the unexpected power peaks of the Kuka robot are detected and used to fire an alert to the user and to the production line. 158
6.23 Overview of the proposed approach. 159
6.24 ISO 10816: Definition of Vibration Severity Levels with respect to the machine class 161
6.25 Definition of severity levels for other observable measurements. In this case, temperature and acoustic emissions are considered. 163
6.26 Overview of a generic Monitoring State Machine (MSM). The states reflect the severity levels defined by ISO-10816. 163
6.27 Overview of a Predictive maintenance supervisor monitoring one MSM (vibration MSM). . . 164
6.28 Overview of a Predictive Maintenance Supervisor (PMS) monitoring two MSMs (vibration and temperature). 165
6.29 Overview of the entire framework in validation mode. 166
6.30 Overview of the transmission system. The gears represent the critical component of the system. The model can be faulted in two different parts: the gear tooth and the vibration sensor. 168

List of Tables

3.1 Critical values at different frequency points for the most significative OPAMP circuit faults and their group. 60

4.1 Verilog-AMS disciplines 65
4.2 Verilog-AMS natures 66
4.3 Mechanical failure mode taxonomy. 67
4.4 Relationship between mechanical and electrical variables in the force-voltage (impedance) analogy. 74
4.5 Relationship between mechanical and electrical variables in the force-current (mobility) analogy. 78
4.6 Mechanical fault taxonomy inferred from the analog fault injection in the equivalent model. . 85
4.7 DC motor with gear train parameters. 89
4.8 The average simulation time and the number of injected faults are reported for different systems. 98
4.9 Physical analogies relationships between mechanical, thermal, and electrical domains. 101

5.1 Simulation results for the half-wave rectifier, the memristor, and the ideal relay. 118
5.2 Main articles related to data-driven behavioral modeling. 120

6.1 Simulation time measurements. 135
6.2 Definition of new standard for temperature MSM. Table I-A reports the experiments for recipes A and B with different tooth fault levels with respect to ISO 10816 severity levels. Table I-b shows the obtained severity level ranges for temperature MSM. 168

List of Listings

3.1 Eldo grammar fragment written in ANTLR. . . . 33
3.2 Spectre grammar fragment written in ANTLR. . . . 33
3.3 Fragment of the Eldo grammar for the component X. . . . 34
3.4 Fragment of the Spectre grammar for the instantiation of a component. . . . 34
3.5 OPAMP subckt described in Eldo language. . . . 37
3.6 OPAMP subckt described in Spectre language. . . . 38
3.7 OPAMP subckt wrapped described in Eldo language. . . . 40
3.8 XML fragment of the OPAMP subcircuit generated through our framework. . . . 41
3.9 Simple netlist modeled with the ELDO language. . . . 42
3.10 Templates for MOSFET defect models. . . . 46
3.11 Custom Defectsim templates for short and open fault. . . . 48
3.12 OPAMP subcircuit modeled in SPECTRE. . . . 50
3.13 OPAMP subcircuit modeled in SPECTRE that contains the transformations in order to be able to use defect injection wrappers. . . . 50
3.14 OPAMP1 testbench written with the SPECTRE language. . . . 50
4.1 RLC circuit modeled in Verilog-AMS. . . . 75
4.2 Double-RLC circuit modeled in Verilog-AMS. . . . 77
4.3 DC motor model, fault-free. . . . 88
4.4 Motor with gear train modeled in Verilog-AMS, fault-free. . . . 90
4.5 Full electrical DC motor model plus wheel, fault-free. . . . 96
4.6 Full electrical DC motor model plus wheel, open failed. . . . 96
4.7 Sketch of SPICE code to control the simulations. . . . 99
4.8 Verilog-AMS modules implementing the DC motor and the Cauer Network module. . . . 104
5.1 Non-linear model of a half-wave rectifier written in Verilog-AMS. . . . 115
5.2 Verilog-AMS PWL model for a half-wave rectifier. . . . 116
5.3 Verilog-AMS top level. . . . 116
5.4 C++ analog branch structure. . . . 116
5.5 C++ abstracted code of half-wave rectifier. Variable name that start with 'b_' are of type *analog_branch_t*. . . . 116
5.6 C++ simulation manager. . . . 117
5.7 Implementation of a simple Wiener-type NL system in Verilog-A. . . . 125
6.1 Original DC motor model, fault-free. . . . 134
A.1 Templates for MOSFET fault models (part 1). . . . 193

A.2 Templates for MOSFET fault models (part 2). 194
A.3 Templates for MOSFET fault models (part 3). 195
B.1 Faulty RLC circuit modeled in Verilog-AMS, open fault. 197
B.2 Faulty RLC circuit modeled in Verilog-AMS, short fault. 198
B.3 Faulty Double-RLC circuit modeled in Verilog-AMS, open fault. 198
B.4 Faulty Double-RLC circuit modeled in Verilog-AMS, short fault. 199

List of Acronyms

AB	AdaBoost
AMQP	Advanced Message Queuing Protocol
AMR	Autonomous Mobile Robot
AMS	Analog and Mixed-Signal
ANTLR	ANother Tool for Language Recognition
AST	Abstract Syntax Tree
COMP	Comparator
CPS	Cyber-Physical System
CUT	Circuit Under Test
DC	Direct Current
DCP	Distributed Co-Simulation Protocol
DIH	Data Integration HUB
DSL	Domain Specification Language
DUT	Device Under Test
EDA	Electronic Design Automation
EMF	Electromotive Force
ERP	Enterprise resource planning
FEM	Finite Element Model
FMEA	Failure Mode and Effect Analysis
FMI	Functional Mock-up Interface
FMU	Functional Mock-up Unit
FSA	Fault Sensitivity Analysis
HDL	Hardware Description Language
IC	Integrated Circuit
ICPS	Industrial Cyber-Physical System
IIOT	Industrial Internet of Things
IP	Intellectual Property
ISO	International Organization of Standardization
KPI	Key Performance Indicator
LPTN	Lumped-Parameter Thermal Network
LTI	Linear Time-Invariant
LU	Lower–Upper

M2M Machine to Machine
MBSE Model-Based Systems Engineering
MEMS Micro-Electro-Mechanical System
MES Manufacturing Execution System
ML Machine Learning
MLP Multilayer Perceptron
MLP-B Bayesian Multilayer Perceptron
MOM Message Oriented Middleware
MOS Metal-Oxide-Semiconductor
MOSFET Metal-Oxide-Semiconductor Field-Effect Transistor
mRMR Minimum Redundancy Maximum Relevance
MSM Monitoring State Machine
MSR Maximum Softmax Response
NR Newton-Raphson
ODE Ordinary Differential Equation
OP Operating Point
OpAmp Operational Amplifier
OPC UA OPC Unified Architecture
PLC Programmable Logic Controller
PMS Predictive Maintenance Supervisor
PWL Piecewise Linear
RF Radio Frequency
RNM Real Number Model
ROM Reduced Order Models
RPC Remote Procedure Call
RTT Round Trip Time
SAW Surface Acoustic Wave
SIL Safety Integrity Levels
SPICE Simulation Program with Integrated Circuit Emphasis
SSL Self-Supervised Learning
SSP System Structure and Parameterization
SVM Support Vector Machine
SysML System Modeling Language
UML Unified Modeling Language
VF Vector Fitting
VP Virtual Platform
VT Voltage Threshold

1

Introduction

1.1 Overview

Industrial companies are undergoing what has been identified as *"a fourth industrial revolution"*, where the technological aspects meet the industrial automation field. In this context of *Industry 4.0*, a factory becomes a complex and heterogeneous ecosystem where many technologies, systems, and workers cooperate to compose a smart factory [2]. This smart factory can be modeled as a composition of Industrial Cyber-Physical System (ICPS) that interact with each other by using different types of physical connections (mechanical, electrical, electromagnetic signal). Workers and machines must cooperate securely, so it is necessary to evaluate and adopt techniques to ensure functional safety in the entire production plant. By looking into the next steps that the *Industry 4.0* should follow, there is a radical change in the vision. If, before, a worker only had to control industrial machinery and ensure its proper functioning by attempting to understand its operations, in the near future, the priority would change. A worker will have to cooperate within the production line with industrial machinery, and as a result, so the worker will have to understand the dynamics of a machine, but also a machine (e.g., collaborative anthropomorphic robot) will have to understand the movements and intentions of a worker to guarantee cooperation between the humans and robots. This vision lays the foundation for the future *Industry 5.0*, where it will be the machinery that will have to adapt its actions to the human factor [3]. These novel propositions come with a series of intellectual and engineering challenges because machinery will have to ensure even higher standards of fairness and safety. In addition, a smart manufacturing system must integrate a very heterogeneous set of components and, at the same time, preserve the functional safety of the whole industrial plant by maintaining resilient production.

To ensure functional safety in a production plant, all the sources of information need to be used to understand the evolution of each machinery. For example, information coming from the Industrial Internet of Things (IIOT) sensors placed everywhere inside a smart factory need to be merged with the production and control information in order to keep the plant monitored, enabling condition monitoring and predictive maintenance techniques [4]. Moreover, information coming from the shop floor needs to be merged with synthetic information coming from the simulation or from different types of algorithms that manage data. When simulation and real data are used to characterize a production plant composed of ICPS, two terms are usually recurrent: digital shadow and digital twin. A digital shadow is an exact replica of the production process, where the data coming from the shop floor are used for the temporal evolution of the twin. While the digital twin is a "virtual" replica of a production process or a replica of a single ICPS machinery that evolves with an internal dynamic that can be synchronized with the real plant. Consequently, by understanding how to add faulty behaviors into the dynamic driving the digital twin evolution is possible to monitor and analyze the evolution of the system in the presence of faulty behaviors to guarantee the functional safety of the entire system.

In the context of industrial digitalization, this thesis has investigated the state-of-the-art to find missing blocks and proposes novel methodologies, techniques, and tools to: model and simulate analog and multi-domain systems expressed at different abstraction levels; abstract low-level models to higher-level behavioral models to be integrated into complex models comprising more physical layers; and monitor industrial systems for automatizing the anomaly detection and predictive maintenance process.

For each macro-area of research addressed in this thesis, the proposed innovative contributions presented in International Conferences and Journals are summarized in Section 7.2.

1.2 Objectives

The thesis proposes new Electronic Design Automation (EDA) methodologies to support the *Industry 4.0* revolution. In this context, all the aspects related to ICPS are suitable to be improved through EDA methodologies [5] because these systems are composed of heterogeneous parts: cyber, physical, and network. The main research motivations that have driven the state-of-the-art improvements proposed in this thesis are based on the following question: *is it possible to create systematically faulty synthetic data from the simulation of ICPS to support the design of systems more resilient that need to preserve the functional safety in every operating condition?* Based on the studies performed during these years, I suggest that the answer is affirmative because it is possible to describe specific fault injection techniques for injecting defects/faults in the different domains composing an ICPS in order to produce faulty temporal series. The concept of defect/fault may have different meanings for different domains; consequently, a clarification on these concepts is proposed in the preamble of Section 2.4.

Starting from this idea, the entire thesis aims to support functional safety in an industrial environment by:

1. Analyze the behaviors of ICPS in the presence of faults:
    a) through the analysis and modeling of analog faults;
    b) through the investigation and modeling of multi-domain faults;
    c) by exploiting the abstraction to obtain faulty models described at different abstraction levels;
    d) by applying these techniques to the industrial field by creating a data fusion infrastructure to correlate real data and models, i.e., the realization of a digital twin.

Figure 1.1 presents a unified view of all the parts involved in the thesis. Starting from the target of this thesis, that is, an ICPS (top left of Figure 1.1), two directions can be followed, modeling or monitoring the system under study. The modeling direction requires that the ICPS under study is modeled through languages or tools at the suitable level of abstraction required to study the system. Then, the modeled system can be simulated to retrieve nominal behaviors that should follow the real behaviors with some delta depending on the precision of the model itself. If more complex analyses are required on the model, it is possible to define systematic alterations of the model by injecting faults. These faults can help to simulate faulty behaviors of the systems that otherwise are difficult to obtain without braking machinery. Moreover, by analyzing the faulty behaviors, fault modes can be identified to support further safety analysis. For different physical domains, fault taxonomies and fault injection techniques are defined to support fault injection campaigns. Also, the identification or non-identification of different faulty behaviors that a system could present under the influence of different faults could depend strictly on the testbench quality, making fundamental testbench qualification approaches. Furthermore, two different methodologies are presented to abstract ICPS models in order to speed up the simulation of the models and allow them to be integrated into complex simulative models. Instead, if the monitoring direction is followed, the data coming from ICPS are

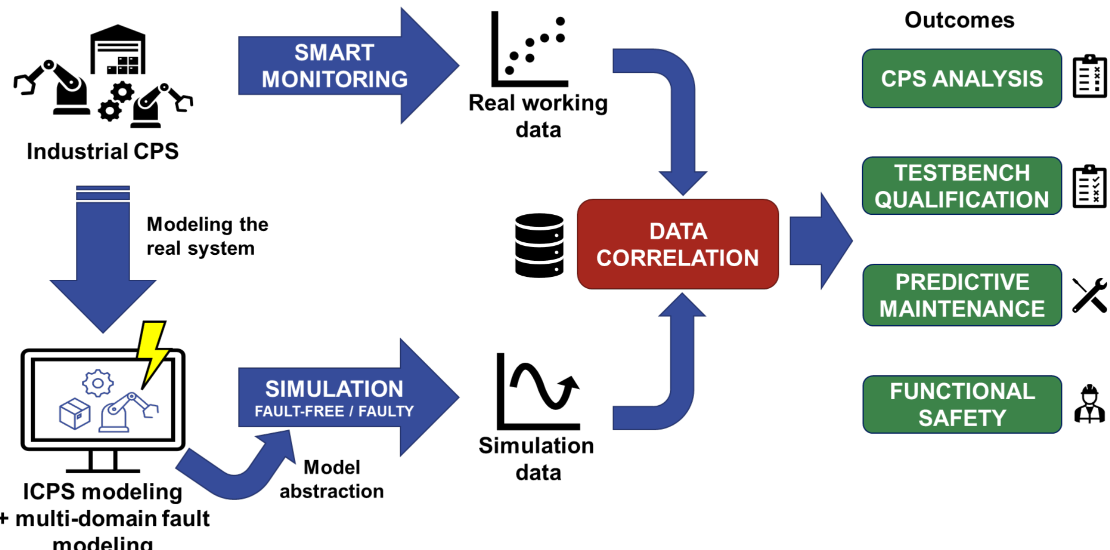


Fig. 1.1: Overview of the proposed methodology, where two flows meet in the middle to produce different outcomes: ICPS analysis, functional safety, predictive maintenance, production optimization.

retrieved through IIOT sensors and processed in order to analyze machine behaviors systematically. Finally, by using data correlation techniques, the real working data and the simulation data (fault-free and faulty) can be merged to increase the quality of further analysis: Cyber-Physical System (CPS) analysis, testbench qualification, predictive maintenance, and functional safety.

1.3 Thesis Outline

Based on the structure of Figure 1.1, the thesis is organized into chapters as follows:

- Chapter 2, introduces the main concepts needed to understand this thesis: ICPS, functional safety, digital twin, and fault analysis.
- Chapter 3, describes the analog fault employed at state of the art; then it proposes two new defect models and new methodologies to inject faults into transistor-level descriptions. Moreover, it analyzes the simulation results through fault grouping techniques in order to identify failure modes. The methodologies analyzed and extended in this chapter are currently state-of-the-art for analog models and are fundamental for this thesis because the same concepts can be applied in other domains that compose an ICPS.
- Chapter 4, describes the multi-domain fault modeling and injection into multi-domain models, e.g., mechanical and electrical systems described through differential equations.
- Chapter 5, proposes two methodologies to abstract analog models to behavioral level models. The first methodology allows abstracting PWL models written in Verilog-A to C++ code. At the same time, the second methodology allows abstracting general analog descriptions (linear and non-linear) through a complex flow that could be applied even to mechanical systems.
- Chapter 6 describe different methodologies to build ICPS models, presents an infrastructure to monitoring a ICPS, and proposes a methodology to correlate synthetic data generated through the simulation with real data coming from a real production line.
- Chapter 7 describes the conclusions and the future research directions.

2

Background

2.1 Industrial Cyber-Physical System (ICPS)

A Cyber-Physical System (CPS) is a system that comprises a digital computation integrated with physical processes, e.g., mechanical actuators. The term "Cyber-Physical System" emerged around 2006 from the research of *Helen Gill* at the National Science Foundation (NSF) in the United States. These systems, by exploiting network communications, are able to monitor and control different physical processes. Usually, the control is implemented with feedback loops where physical machinery behaviors affect computations and vice versa. A CPS is defined and built as the intersection of the physical and the cyber part and not as a union of different technologies. It is not sufficient to understand the behavior of physical components and the computational components separately; instead, it is necessary to understand their interactions. To design such systems is required to understand the joint dynamics of software, networks, and physical processes [6].

In the latest years, the general term CPS is used in the industrial context as Industrial Cyber-Physical System (ICPS). In general, a CPS could be a general system, while a ICPS is a CPS envisioned specifically for an industrial context with specific constraints (see Figure 2.1 for the representation of some ICPS).

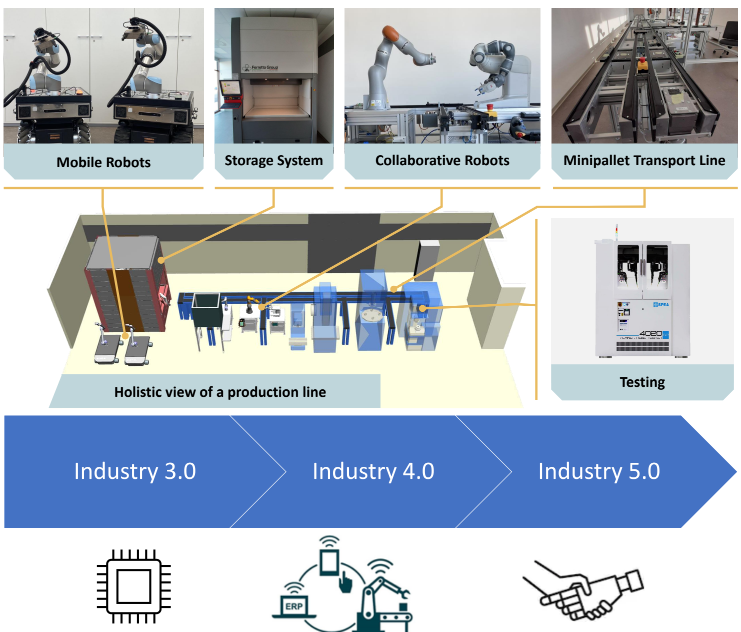


Fig. 2.1: Evolution of the *Industry* from 2000 (Industry 3.0) to nowadays (Industry 5.0).

Figure 2.1 presents a holistic view of a production line (exemplified in the Industrial Computer Engineering Laboratory (ICELab) of the University of Verona) in which different types of machinery (ICPS), collaborative or autonomous, composes the pilot production line following the dictates of the Industry 4.0. Starting from 2010, the "fourth industrial revolution" has mixed different technologies, such as ICPS, Industrial Internet of Things (IIOT), networking, and machine learning. In this vision, only machines are allowed in the production process. Now, the focus moves from Industry 4.0 to Industry 5.0, where the machines composing the production plant and human intelligence are mixed together. In this context, guaranteeing the functional safety of an entire production plant (composed by interconnected ICPS) became fundamental, especially for worker safety.

2.2 Functional safety

The industry sector is involved in a fast expansion period and transition to new technologies, particularly in the car sector, which is changing the way cars are designed, used, and sold. In the automotive sector, driver safety technologies, traffic congestion, and environmental concerns are factors that strongly influence the next generation of connected vehicles [7]. All these internal and environmental factors that influence a connected car are related to different classes of CPSs that are built to improve powertrain emissions, enhance safety and provide connectivity to other cars and road infrastructure by using telecommunication networks. However, these sophisticated systems require a fool-proof way to preserve the driver's safety, which is called *functional safety*. Managing the need for functional safety for different CPS is essential for success in all industrial fields. However, it presents challenges to even the most experienced semiconductor vendors. Moreover, ensuring the functional safety of CPS composed of digital, analog, and physical parts requires building new methodologies and standards able to consider the functional safety in a holistic way for all the aspects of a CPS.

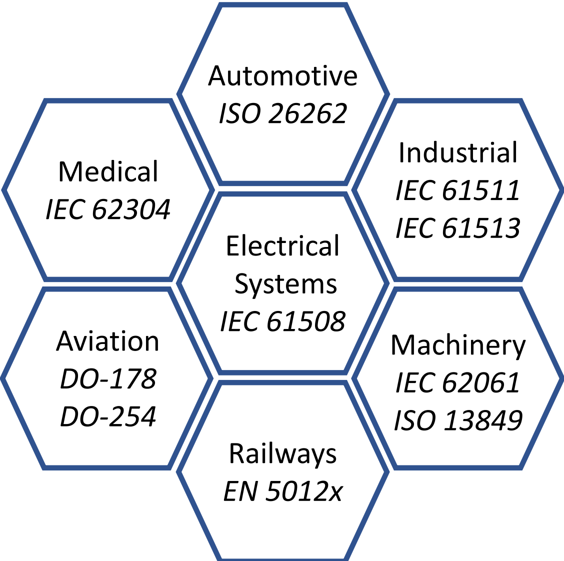


Fig. 2.2: Functional safety standards subdivided for the different industrial sectors.

Functional safety means ensuring the safe operation of systems even when something goes wrong [8]. Each industry typically has a standard to guide developments and define the minimum set of actions necessary to guarantee safety. For automotive electronics the ISO 26262 standard [9] defines the functional

safety as: *the absence of any risk caused by malfunctioning of electrical components*. Functional safety is moving away from a specialist requirement to become normality as the number of complex applications that rely on electronics is increasing, ranging from automotive to medical and industrial devices. Different safety standards exist for other industrial sectors (see Figure 2.2): IEC 61508 standard [10] for electrical and electronic systems, DO-178 [11], and DO-254 [12] standards for airborne electronic hardware, IEC 62278 [13] for railways, IEC 62061 [14], and ISO 13849 [15] standards for machinery, IEC 61511 [16], and IEC 61513 [17] standards for industrial plants and manufacturing processes, and IEC 62304 [18] standard for medical applications.

All these standards listed have their own definitions, although more importantly, they also set their own terminology and guidance for engineering developments, including target metrics applicable to silicon Intellectual Property (IP). In practice, it is possible to address the needs of multiple standards by identifying their specific requirements and adopting common principles such as quality management and a focus on safety from the outset.

In detail, functional safety means a system that is demonstrably safe in accordance with the target standards specific to a machinery class. Safety requires predictable failure modes, which could be with full functionality, graceful degradation in functionality, or a clean shutdown followed by a reset and restart of the system. Not all faults will lead to hazardous events immediately, but they could degrade the system's performance over time. For example, a fault in a car's power steering might lead to incorrect sudden steering action. However, since the electronic and mechanical designs will have natural timing delays, faults can be tolerated for a specific amount of time, often many milliseconds or more. In the ISO 26262 standard, this time is known as the fault-tolerant time interval and depends on the potential hazardous event and the system design. Theoretically, for an application that is more safety-critical, more unsafe faults must be mitigated. In an ideal world, functional safety would have no impact on system performance, although, in practice, many of the countermeasures available to designers can significantly affect system performance. So one of the challenges is to achieve sufficient functional safety whilst trying not to adversely affect the system or its cost to design or manufacture.

The ISO 26262 standard outlines the requirements for fault injection testing and defines metrics to evaluate the effectiveness of the testing for different Safety Integrity Levels (SIL). These measures ensure that the fault injection testing is comprehensive and that the system can handle a range of potential failures. Fault injection is an important aspect of safety engineering in many industrial applications, as it helps to ensure the reliability and safety of electronic systems in vehicles. These techniques are used to test the robustness of electronic systems by intentionally introducing faults or errors into the system. The level of safety integrity required for an electronic system is determined by its criticality function, with systems controlling safety-critical functions such as brakes and airbags requiring a higher level of integrity than those controlling less critical functions. The ISO 26262 standard provides guidelines on the level of integrity required and the metrics to be used to quantify the integrity of systems [19]. Based on these problematics well documented for the automotive sectors, this thesis proposes different methodologies to generate faulty behaviors for different classes of ICPS based on multi-domain faults.

2.3 Digital twin

Digital twins of production lines aim at connecting the physical manufacturing space with its virtual representation to improve the production process [20] and to reinforce the modules that need to preserve functional safety in an industrial context [21]. The virtual part records the historical evolution of the physical plant and predicts its evolution with the goal of identifying malfunctioning and possible optimizations.

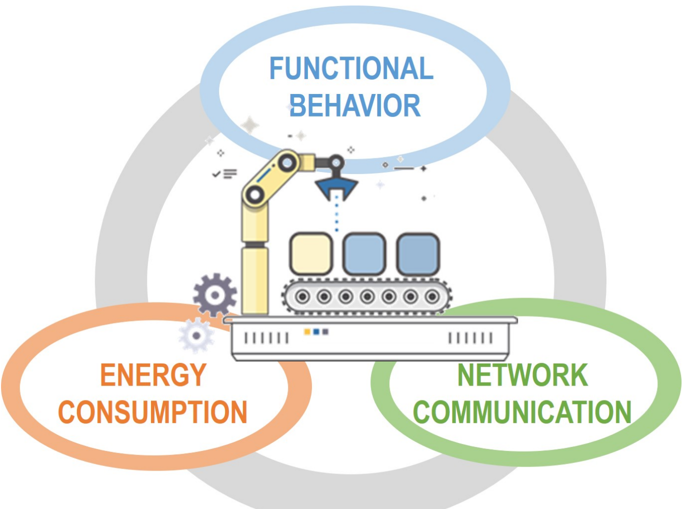


Fig. 2.3: Physical dimensions that can be considered for the digital twin construction: functional behavior, energy consumption, and network communication.

Meanwhile, the physical part provides sensed data and behaviors to allow a continuous refinement and calibration of the virtual part. The role of the digital twin in the lifetime of a production line is crucial at different stages [22, 23]:

- it acts as a *virtual prototype* of the production line or a single machinery to evaluate its behavior before its actual implementation;
- it enables *effective decision making* to make informed data-driven decisions and reduce possible sources of inefficiencies;
- it *monitors line operation at run time* and predicts its behavior, thus providing a reference model of its evolution for failure detection and possible optimizations.

During the construction of the digital twin of a production line, the focus is mostly on its *functional behavior*, i.e., on the interaction between humans and equipment, on the movement of items on the production line, and on the manipulation of items by robots and machinery. Monitoring is indeed relevant to keep track of the evolution of the production line [24, 25].

However, this is only one aspect of the problem: other dimensions, hereby called *extra-functional*, can be considered with positive benefits on the effectiveness and efficiency of the production process. Figure 2.3 depicts this concept by showing two commonly used extra-functional dimensions: namely, energy consumption and communication.

Energy consumption monitoring, prediction, and optimization are important to improve the cost-effectiveness of a production line, to meet the emission goals, and also to monitor the state of health of the equipment. Power models can be used both at design and configuration time to identify efficient production recipes and to optimize energy waste [26, 27]. Additionally, power models included in the virtual part of the digital twin are extremely useful at run time, to detect any gradual increase of energy consumption as an effect of degradation of the equipment components, to detect the health of the production line, and to schedule maintenance operations [28].

Another important dimension is *networking and communication*. Today's factory machines are ever more connected with internal and external systems to the factory. Communications have different types of requirements: control communications at the lowest level are susceptible to delays and errors, while monitoring data to be used by machine learning procedures require large network capacity but with not

particularly short latency constraints. The spatial features (i.e., walls and distances) may block communications or impact on cabling costs. No automatic technique is currently available in the context of Industry 4.0 to choose the best mix of (wired and wireless) network architectures according to spacial constraints, cost, and quality-of-service requirements.

Furthermore, by simulating a digital twin extended with different classes of faults, it is possible to understand possible failures and define strategies to prevent them by following a “how-to” strategy to anticipate possible disruptive failures. Moreover, by exploiting such techniques, it is possible to support the definition of a complete and tested Failure Mode and Effect Analysis (FMEA). This process could also identify possible critical paths that cause failures and enable the designers to design CPS more resilient to faults or predict “what-next” through the simulation, enabling alternative scenarios, robustness analysis, and further optimizations.

2.3.1 Maintenance process

The production processes within the manufacturing companies must be continuously improved to increase efficiency and remain competitive with other companies. In order to maintain competitiveness within the market, manufacturing companies must increasingly concentrate on high-volume production while reducing profit margins. In this scenario, an effective maintenance plan and the ability to react before problems lead to sudden production downtime are strategic. This can be achieved by exploiting the potentialities of a digital twin of an ICPS.

Maintenance, as defined in the Standard UNI EN 13306:2010, is a combination of all technical, administrative, and managerial actions during the life cycle of a machine designed to maintain or restore it to a correct state. Maintenance can be categorized into corrective and preventive maintenance. *Condition Based Maintenance* is currently one of the most effective policies adopted by industries. Concerning the UNI EN 13306 standard, preventive maintenance is based on monitoring several parameters of the machine, which reflect its state of health. However, the trend in recent years has moved to *Predictive Maintenance* as described in recent surveys [29]. This change is closely related to the digital revolution, which is affecting small and medium-sized enterprises. Industry 4.0, with the development of the IIoT and standardized communication protocols such as OPC UA [30], allows companies to perform advanced analysis and create a digital twin of the production line. In this context, predictive maintenance is an advanced tool that allows companies to carry out analysis on the field level. Furthermore, Predictive Maintenance can decrease the risk of catastrophic failure and prolong apparatus aging. This was previously much more complex due to non-standardized communication protocols. A proper application of Predictive Maintenance can reduce maintenance costs and also enhance system reliability. These maintenance techniques enable companies to analyze variables that are Key Performance Indicators (KPIs) for machine efficiency. This information is correlated with equipment maintenance records to predict which equipment may suddenly require unscheduled maintenance, thus preventing a fault.

The UNI EN 13306/2010 standard provides precise maintenance types definitions [31]:

- *maintenance* (see UNI 13306 p. 3.1): a combination of all technical, administrative, and management actions during the life cycle of a system, designed to maintain or restore it to a state in which it can perform the required function;
- *preventive maintenance* (see UNI 13306 p. 7.1): maintenance carried out at predetermined intervals or in accordance with prescribed criteria and provided for in order to reduce the probability of failure or degradation of the system;

- *cyclic maintenance* (see UNI 13306 p. 7.2): preventive maintenance carried out according to specified time intervals, but without a prior investigation of the system's condition;
- *condition based maintenance* (see UNI 13306 p.7.3): is part of preventive maintenance, includes a combination of condition monitoring and/or inspection and/or testing, and the necessary maintenance actions;
- *predictive maintenance* (see UNI 13306 p. 7.4): is part of preventive maintenance, performed on the basis of a prediction derived from repeated analysis or from the evaluation of significant parameters relating to the degradation of the system;
- *corrective maintenance* (see UNI 13306 p. 7.5): maintenance performed after a failure has been detected to restore a system to full functionality.

2.4 Fault modeling, injection, and simulation

An interconnection of ICPS is the core of a smart manufacturing plant, as previously described in this chapter. For each ICPS is thus possible to exploit model-based approaches to build a *virtual replica* of the machinery named digital twin. Alternatively, a data-driven solution can be used to derive the digital twin. In the first case, by exploiting the potentialities of model-based solutions is possible to describe models that combine different physical layers. These layers composing the digital twin can comprise different domains, e.g., cyber, physical, and the network. Each layer usually is modeled through specific languages or tools used in engineering applications. By simulating these models is easy to derive nominal traces that represents the standard behavior of that machine. When faulty traces are required for advanced analyses, usually, these are not available from the physical world; thus, producing them in a synthetic way through the simulation became strategic.

In this context, fault modeling and injection techniques play a fundamental role, enabling the generation of faulty traces through the simulation. Fault injection techniques are state-of-the-art for the digital domain, a necessary step to certify each type of critical device as specified in the ISO 26262 standard [9]. Through fault injection techniques is possible to verify if fault-tolerant architectures are sufficiently resilient in the presence of faults. In other domains, e.g., in the analog domain, the need for sound defect modeling and injection techniques is becoming relevant because the complexity of the systems is continuously growing. A clarification related to the distinction between defect and fault is provided in the following paragraph. The same problem arises when there is a need to inject faults into other physical domains, e.g., the mechanical or the thermal domain. Only specific solutions for some specific classes of machinery exist to model and inject faults into these models. In this scenario, a digital twin composed of different physical layers instrumented with different classes of faults can produce enormous faulty traces, allowing to design and testing of safety mechanisms and preserving the overall functional safety of a system.

Consequently, creating universal fault models for different physical layers and automatic methodologies for injecting and simulating these faults became strategic. This thesis, starting from the analog domain, proposes new techniques to improve the state-of-the-art related to fault modeling, injection, and simulation.

Clarification about the distinction between defect and fault

For the digital domain, the standard ISO 26262 [9] defines the guidelines for implementing a fault injection campaign to ensure the functional safety of road vehicles. With the focus on digital circuits, in literature, these methodologies are referred as fault modeling and injection techniques because the functionality of a digital circuit (fault-free and faulty) can be seen as an abstraction of the analog circuit that implements the

physical functionality. Instead, when referring to the analog domain, the correct term used to identify these techniques changes from fault to defect. Thus defect models are characterized to mimic physical deformations and injected into a netlist can produce a fault after the simulation. Due to these historical reasons, the industry uses fault injection and simulation more than defect injection and simulation when referring to the analog testing field. This error is probably related to the fact that the first attempt to standardize the terminology in this field is proposed by the IEEE P2427 draft standard [32]. Previously, in some cases, these two terms were swapped and used without distinguishing between their different significance in the digital and analog domains. Consequently, by analyzing the literature terminology and maintaining compliance with the P2427 draft standard in this thesis, the terms fault and defect are used with different meanings in the various chapters. Specifically, when referring to the analog field, the terms used are defect injection and fault simulation. While, when referring to more abstract faults defined at the behavioral level for different physical domains as in Chapter 4, the terms used are fault injection and fault simulation. Furthermore, a defect/fault is solely localized on the internal structure of a design. While a failure mode is a faulty behavior that can affect a design instrumented with a defect/fault.

3

Analog fault modeling

The *functional safety* of critical applications like automotive and aerospace is of primary concern in today's world for an Integrated Circuit (IC) designer. In this context, analog defect modeling is still an open research topic [33, 34]. In the recent past, various analog defect modeling techniques for fault injection have been proposed to evaluate the safety of critical systems. The main commonly used defect models in analog circuits are opens and shorts, modeled through different parameter values of a resistance component. Despite the ongoing work to standardize the definition of these defect models and coverage calculation methods in the IEEE P2427 draft standard [32], there is a lack of a unified and portable method to define defect templates that can be injected in analog circuits to simulate faulty behaviors.

Generally, each of the existing Electronic Design Automation (EDA) tool sets for fault injection proposes its own proprietary method to specify how defects should be modeled and injected into a design. This chapter proposes different methodologies built to help designers to test faulty behaviors in their analog designs and is organized as follows:

- Section 3.1 presents the available analog defect modeling approaches enriched with the proposal of new stuck-on/off defect models;
- Section 3.2 proposes an overview of the main state-of-the-art techniques related to fault injection and simulation in analog circuits describing the main limitations of the current techniques;
- Section 3.3 describes the proposal of a new framework for the manipulation of the most common languages used to describe transistor-level netlists: such as Spice, Eldo, and Spectre;
- Section 3.4 proposes a new implementation of generic analog defect templates portables across different analog simulators based on the Verilog-A language;
- Section 3.5 describes new techniques proposed to reduce the computational effort of an entire fault injection campaign by exploiting fault grouping techniques based on the equivalence of different fault behaviors.

The terminology used throughout this chapter comprises:

- **Defect**: unwanted physical change in a circuit element or connection between circuit elements that are not within fabrication specifications for the circuit element or connection;
- **Fault**: a model of the impact of a defect at component level in a certain discipline, e.g., a drain/source short, a transistor stuck on/off;
- **Failure mode**: deviant behavior of a subsystem that may cause the system to fail to execute its intended function, e.g., an operational amplifier output that oscillates or that has a voltage or current offset deviating unacceptably from the fault-free behavior;
- **Fault-site**: each component instance that can be affected;

- **Fault coverage**: percentage of fault detected during the defect injection campaign.

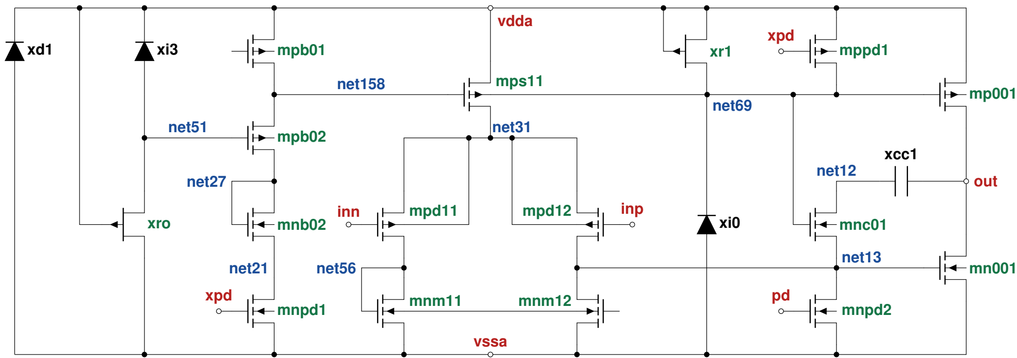


Fig. 3.1: Transistor-level description of the OPAMP model extracted from the analog benchmarks suite.

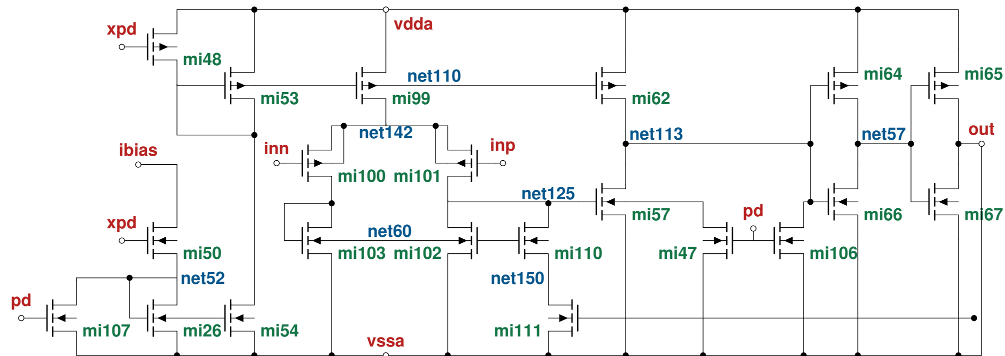


Fig. 3.2: Transistor-level description of the comparator model extracted from the analog benchmarks suite.

Analog Benchmarks suite:

All the proposed methodologies have been validated on the circuits of the Analog Benchmarks suite made available by the P2427 working group [35]. This A/MS Benchmarks suite comprises different circuits, e.g., an Operational Amplifier (OpAmp) shown in Figure 3.1, and a comparator shown in Figure 3.2. These analog circuits are generally used as a benchmark to test new methodologies because more complex circuits are not available in open-source format.

3.1 Analog defect modeling: investigation on realistic stuck-on/off defects

Recent advances in the modeling of MOSFET gate defects suggest a more physical approach [36, 37, 38]. They consider an excess of fixed charges trapped in the gate-oxide layer or an excess of interface traps at its boundary as the cause of intrinsic faults in MOS transistors. The literature suggests that the impact of these new defect models on analog circuits has not yet been tested but only theorized. The impact of a fault on an analog circuit is how the circuit behavior changes in relation to a specific defect injected; that is, the effect on the circuit. This section presents an implementation of the new physically-inspired MOS defect models and investigates their effects on analog circuits.

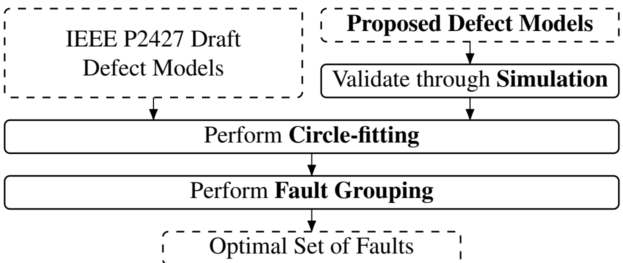


Fig. 3.3: Overview of the entire workflow.

Figure 3.3 shows the proposed workflow. We start by considering the entire set of *hard* defects as proposed by the IEEE P2427 draft standard [32], then an extension of this fault set with new *parametric* defect models that represent transistors stuck-on or stuck-off behaviors. The new defect models are simplified implementations of a physical model accounting for excess oxide charge and excess interface traps. The faulty circuit-level behaviors induced by the new defect models are compared with those induced by the hard defect models inspired by the P2427 draft standard using the transient simulations. In order to compare the impact of the injected defects in a generic fashion and to ensure that conclusions remain valid for the circuits under test used in other environments, albeit roughly under the same biasing conditions, we extract *AC* matrices at multiple frequencies and various dynamic operating points during the transient simulations and compare the obtained matrices for the various injected defects. Applying similarity measures to the collection of *AC* matrices, defects inducing similar faulty behaviors can be grouped together, and one representative defect can be elected for each class of similar induced faulty behaviors.

The main contributions of this work are:

- Proposal of new physical stuck-on/off defect models described at transistor level;
- Analyzing the behavior of the proposed defects on real transistor-level circuits;
- Comparing the proposed defects with the IEEE P2427 defects using the transient analysis;

3.1.1 State of the art and definitions

This section first explains the basics of analog defects defined at the transistor level.

Transistor-level defects

The IEEE P2427 draft standard proposes to distinguish between *hard* and *parametric* defects, the latter being associated with a progressive parametric deviation and considered to be more difficult to detect. *Hard* are defects considered to cause a permanent change in the circuit's topology and are often associated with the application of basic *short* or *open* defects. Figure 3.4 proposes the locations of possible analog defects of a transistor-level description of an inverter circuit: open, short/bridge, parametric, and pulse. For MOS transistors, *hard* defects may be further characterized by their effect on the main current path between the transistor's anode (drain) and cathode (source) and split accordingly into:

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

Interestingly, there is still an ongoing debate whether transistor *stuck-on* or *stuck-off* defects as generated by excess oxide charges or interface traps should be considered *hard* or *parametric* defects. In particular, one

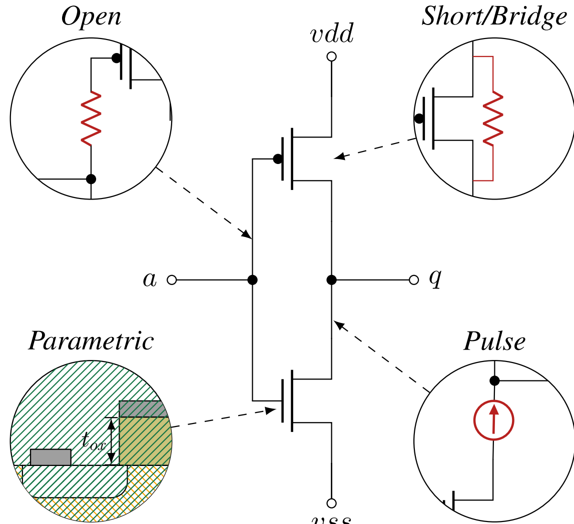


Fig. 3.4: Locations of possible analog defects of a transistor-level description of an inverter.

may wonder whether transistor fault models generated by the application of basic *short* or *open* defects may "cover" the behavior induced by excess oxide charges or interface traps in MOS transistors as physically described in [36]. Considering that *stuck-on* or *stuck-off* transistor defects due to excess oxide charges or interface traps are defects that, unlike the *hard* defects listed previously, change the intrinsic characteristics of the transistor, rather than modifying its connectivity to the circuit, one may already suspect that their induced faulty behavior will be "specific" or "distinguishable" from defects induced by the application of generic *opens* and *shorts*.

3.1.2 Proposed stuck-on/off defect models

Our present investigation is driven by the intuition that none of the *hard* defects as envisioned by the IEEE P2427 draft standard seem to naturally "cover" the transistor *stuck-on* or *stuck-off* behaviors which are known to occur in MOSFET's and are caused by excess trapped charge in the gate oxide or excess interface states [36]. With analog circuits in mind, we focus our study on physically-inspired defect models, which render the effect on transistor characteristics of excess gate-oxide charges or excess interface traps in a realistic fashion. As illustrated in figure 3.5, *trapped charges*, marked in red "○", are fixed positive charges which are trapped in the gate oxide and have a primary effect a shifting of the threshold voltage. *Interface traps*, represented with red "x", are parasitic states which can capture and release minority carriers normally intended to populate the channel but which get blocked in the trap and can not participate in the current conduction in the channel anymore. Traps reduce the efficiency of the MOS gate to attract carriers into the channel for conduction. They manifest themselves as a reduction of the sub-threshold slope and a shift of

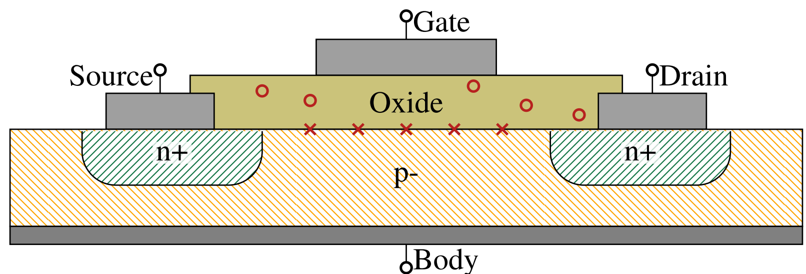


Fig. 3.5: MOS cross-section with oxide trapped charges and interface traps.

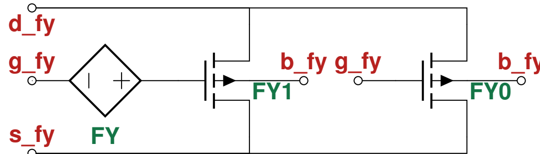


Fig. 3.6: Proposed MOS stuck-on defect model.

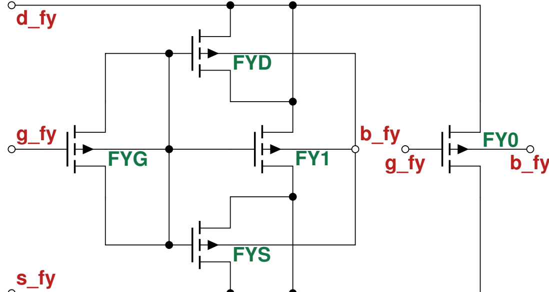


Fig. 3.7: Proposed MOS stuck-off defect model.

the threshold voltage. Esqueda and Barnaby [36] chose to model the effects of excess fixed oxide charges and excess interface states in the form of a bias-dependent voltage source placed in series with the gate of the original transistor model. However, the resulting model references many parameters typically used in MOS transistor models and is not particularly appropriate for use in automatic fault injection templates. We have adopted an alternative but equivalent approach based on the equivalent circuits shown in figures 3.6 and 3.7.

The equivalent circuit shown in figure 3.6 models an excess of oxide-trapped charges. These charges being generally of positive type, have the following effects on MOS transistors :

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

The equivalent circuit shown in figure 3.7 models an excess of interface traps. A capacitive divider with `FYG` is used to reduce the voltage swing by half on the gate electrode of `FY1`, mimicking the efficiency reduction caused by the interface states. In order to maintain the overall capacitance values at the original level, dummy MOS capacitors and `FYS` and `FYD` are added. The effects of these additions on the MOS transistor model are as follows:

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

The equivalent circuits of Figure 3.6 and Figure 3.7 further also contain a fault-free instance labeled `FY0`. This instance is used in conjunction with the faulty `FY1` instance to allow modulation of the defect model by a fault parameter `FP`, which ranges between 1 and 0. The defect modulation is achieved by specifying a multiplier $M_{Y1} = FP$ on the faulty `FY1` instance and a multiplier $M_{Y0} = (1 - FP)$ on the fault-free instance, realizing, in effect a convex interpolation between the two cases. The parameter `FP` represents the fraction of the MOSFET affected by the defect and will be used here to generate fault circles (see Section 3.5).

Figure 3.8 shows the PMOS ID(VG) characteristics for the fault-free, the standard MOS hard-on/off, and the proposed stuck-on/off. While Figure 3.9 shows the NMOS ID(VG) characteristics for the fault-free, the standard MOS hard-on/off, and for the proposed stuck-on/off. The defect for excess fixed oxide

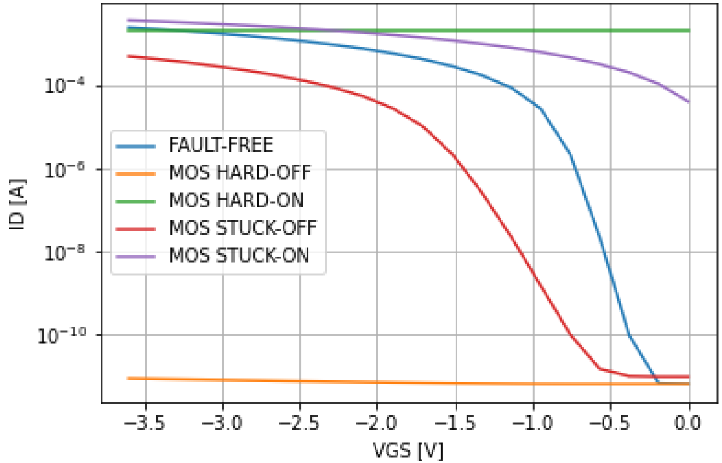


Fig. 3.8: PMOS ID(VG) characteristics for fault-free, hard-on/off, and stuck-on/off.

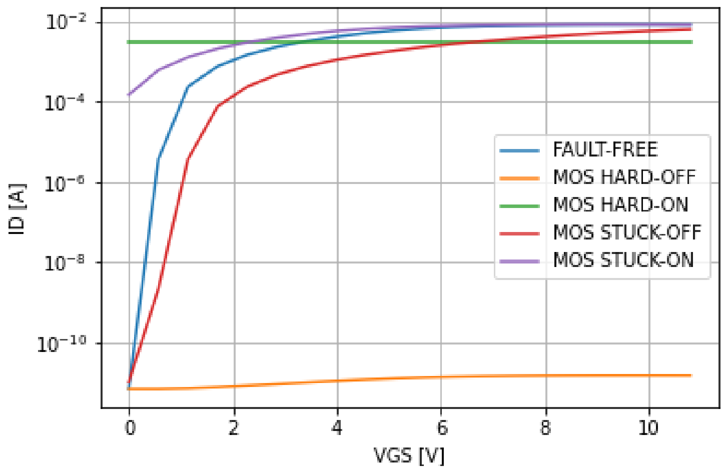


Fig. 3.9: NMOS ID(VG) characteristics for fault-free, hard-on/off, and stuck-on/off.

charges is labeled "*MOS stuck-on*", as this is the behavior that most closely approximates the effect of those charges. The defect corresponding to excess interface charges is labeled as "*MOS stuck-off*" as the current, in this case, is substantially (10x) lower than that of the fault-free case. The ID(VG) characteristics for the proposed stuck-on/off appear more realistic with respect to standard hard-on/off behavior.

The Figure 3.8 and Figure 3.9 further also present two alternatives "naive behavioral" implementations of MOSFETs defects:

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

After this analysis of the ID(VG) characteristic of the representative faults, the results waveforms are compared to identify similar and different behavior between each fault.

3.1.3 Experimental validation

In this section, the proposed defect models are compared with the entire set of defects proposed in the IEEE P2427 draft standard [32]. This comparison is based on the simulation of real circuits, an OpAmp,

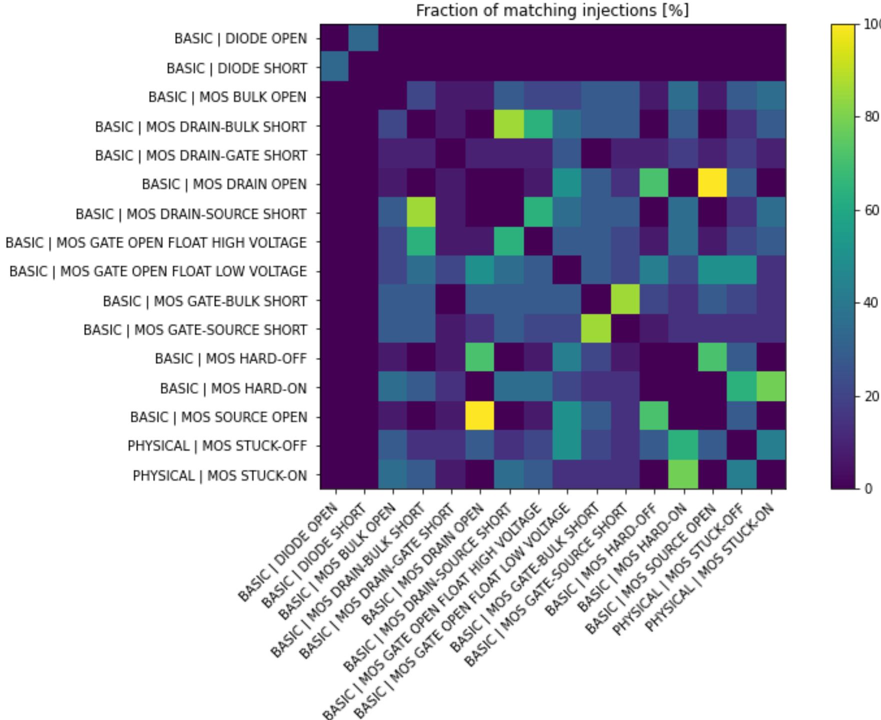


Fig. 3.10: Fault equivalence plot relative to the P2427 defects and the proposed stuck-on/off defect models for the OpAmp circuit.

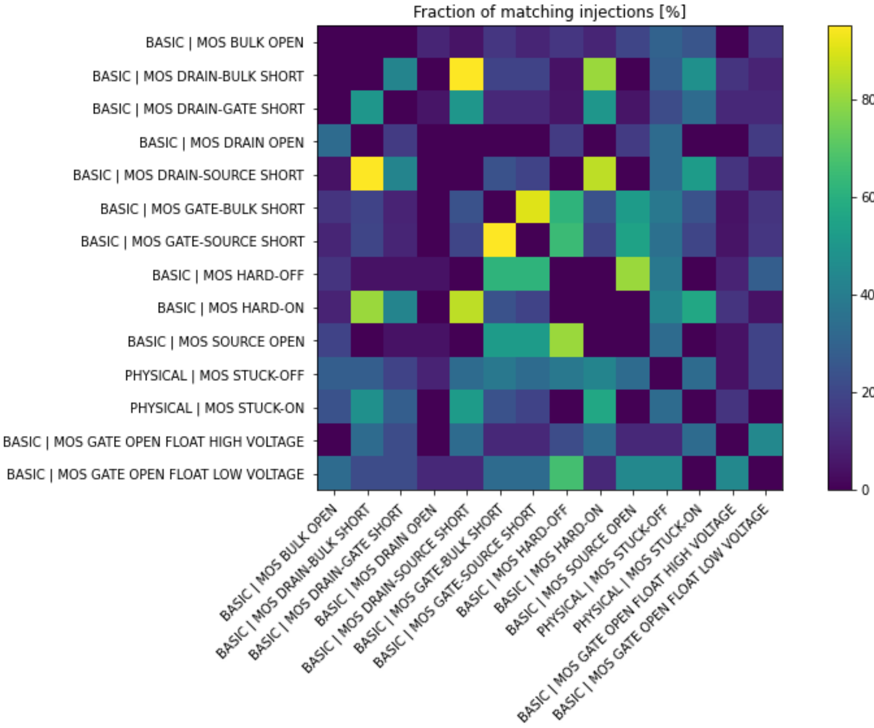


Fig. 3.11: Fault equivalence results between P2427 defects and the proposed stuck-on/off defect models for the Comparator (COMP) circuit.

and a COMP, taken from the analog benchmarks suite [35]. The presented fault grouping method is tested on these circuits to group faults with equivalent frequency behavior. The simulations are performed with a SPICE-like commercial simulator.

Analog circuits analyzed

The proposed workflow is validated by using transistor-level descriptions extracted from the analog-benchmark circuits collection [35]. In particular, the Circuit Under Tests (CUT) is an operational amplifier (shown in Figure 3.1) and a comparator (shown in Figure 3.2). Principal circuit elements of the COMP and the OpAmp are NMOS and PMOS transistors, diodes, JFET transistors, and capacitors. In these transistor-level descriptions, many defect-site locations are possible. All the defects proposed in the P2427 draft standard, plus two new stuck-on/off defect models, are used to validate the workflow. In particular, the defects considered are hard defect models as specified in IEEE P2427 draft standard, e.g., one open for a resistor (R), capacitor (C), and the inductor (L); two opens at the gate and drain for a MOS; a short between terminal pairs unless the nodes are already tied together in the design.

Transient analysis on the proposed stuck-on/off defects

The proposed stuck-on/off defects described in Section 3.1.2 is compared with the defects proposed in the IEEE P2427 draft standard. To compare the faults, a transient analysis is performed for each fault. The results are plotted in Figure 3.11 for the COMP and Figure 3.10 for the OpAmp circuit. In the plots, matching close to 100% (yellow) means that the fault type on the row was in the same group as the fault type on the column nearly all the time for all instances in which it was injected in the reference circuit. In Figure 3.11, it looks like for some faults; there are more than one faults candidate that is equivalent with respect to Figure 3.10. From these graphs, it is easy to see how standard defects (defined in the IEEE P2427 draft standard) cannot produce the same transient behaviors generated by the proposed stuck-on/off defects.

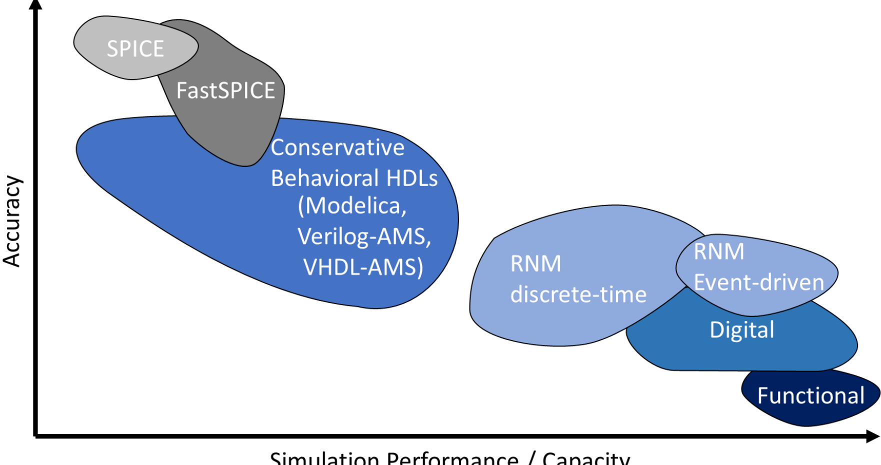


Fig. 3.12: Simulation performance/accuracy of different modeling techniques.

3.2 Analog defect injection and fault simulation techniques

Nowadays, with the increase in the complexity of analog and mixed-signal circuits, guaranteeing the functional safety of both digital and analog circuits is fundamental to reducing every risk of failure in Cyber-Physical System (CPS) and Industrial Cyber-Physical System (ICPS) [9]. Building efficient analog defect injection and fault simulation [34] techniques became strategic for manufacturers. In this context, the maturity of the techniques used in the analog field is lagging behind compared with the digital field. Simulation Program with Integrated Circuit Emphasis (SPICE)-based simulators are still the core technology to simulate analog circuits (fault-free and faulty) described at the transistor level [39]. With the SPICE family software is possible to predict the behavior of electronic circuits by describing them as a netlist. With this software is possible to simulate electronic components ranging from passive elements, e.g., resistors and capacitors, to complex semiconductor modules, e.g., Metal-Oxide-Semiconductor Field-Effect Transistors (MOSFETs). Each electrical component included in the SPICE libraries is associated with a mathematical representation. Then, the entire set of equations related to each component present in the netlist is used to solve the circuit by using Kirchhoff's laws. In particular, SPICE uses a modified nodal solver and exploits the current law: the sum of the currents into each node is zero [40]. When simulating a netlist is necessary to find a good tradeoff between simulation speed and accuracy. By varying the simulation settings is possible to change the level of accuracy needed for a specific simulation. Consequently, for each design is necessary to find the optimal accuracy required for the results at the cost of the speed of the simulation. Then, the simulation can be related to the fault injection process by correctly instrumenting the original netlist with the defects.

A netlist can also be described by other languages, e.g., Hardware Description Languages (HDLs) with an equivalent behavior but more abstracted. Figure 3.12 shows the balance between simulation performance and accuracy that can be achieved by exploiting different modeling techniques. For example, the family of SPICE languages allows for greater accuracy but requires more time to accomplish simulations, while the functional simulation is much faster but less accurate. However, the circuits are becoming more complex as the number of transistors increases (see the reference circuit in the analog-benchmark suite [35]); thus, more evolved methodologies are required. Simulating an analog circuit for a set of input stimuli can require a lot of time and could vary from a fraction of a second to several days depending on the complexity and details accuracy of the Circuit Under Test (CUT). The complexity of a defect injection campaign is directly proportional to the number of defects to be simulated because, for each injected defect, a complete simulation using the nominal input stimuli needs to be computed to retrieve faulty data/matrices.

In the literature, researchers from industry and academia suggested different techniques to reduce the overall simulation time required to simulate an entire set of defects. These techniques include parallel simulation on different cores [41], analysis in the frequency domain at specific operating points [42], usage of high-level models [43], Monte Carlo-based simulations [44], or fault sensitivity analysis [45]. Another direction exploited is to reduce the set of defects that need to be simulated, e.g., with fault grouping techniques. The defect models injected at transistor level are different for each component, e.g., a fault for a MOSFET transistor [46] could be a stuck-on between drain-source, a stuck-off between drain-gate, and so forth. The first attempt to standardize the defect models for each component in a transistor-level description will be released with the standard IEEE P2427 [32], currently in a draft version. This section aims to present a systematic review that explores all the techniques proposed in the literature on analog defect injection and fault simulation. A significant literature review is available on digital fault injection [47]. The digital fault injection approaches are divided into three approaches: hardware-based (physical), simulation-based, and emulation-based. On the contrary, an analogous review of analog defect injection and simulation techniques

is not presented in the literature. In this section, the most important techniques for defect injection and fault simulation in analog circuits are analyzed in detail by showing their advantages and limitations.

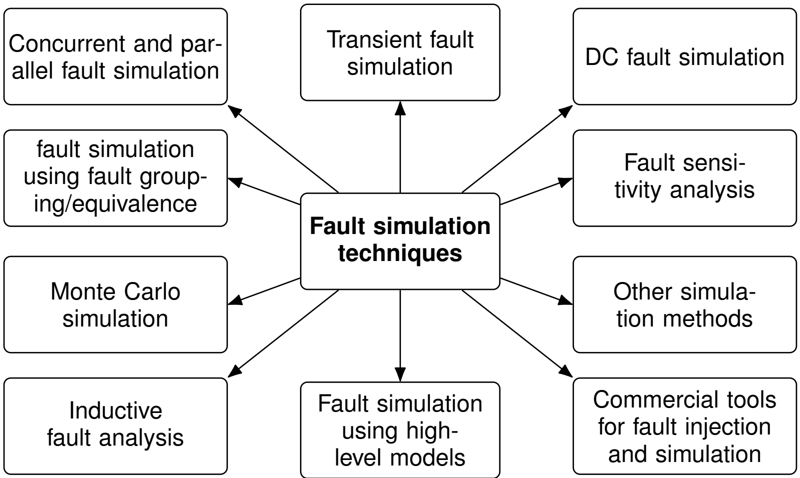


Fig. 3.13: Fault simulation techniques.

3.2.1 State of the art techniques in defect injection and fault simulation

Figure 3.13 describes the main categories of defect injection and fault simulation techniques: transient fault simulation, DC fault simulation, fault sensitivity analysis, fault simulation using high-level models, inductive fault analysis, Monte Carlo simulation, fault simulation using fault grouping/equivalence, and concurrent and parallel fault simulation.

Transient fault simulation

In analog circuits, usually, defects are injected at the transistor level and simulated with SPICE-based simulators. SPICE-based simulation is time-consuming because it allows accurate simulations of transistor-level models. Consequently, the overall simulation time is considerably increased, and furthermore, the reliability of each simulation of a faulty circuit is often low due to convergence issues of the solver. A common problem encountered with transistor-level defect injection is that the defect parameter value required to model the defect is unknown. This value is referred as the parameter value of a defect, and its value should correspond to the common parameter value of a defect. In the literature, many authors proposed different values for the injection. The IEEE P2427 draft standard provides hints about the range of values suitable to model the parameter value for short: $0-200k\Omega$, and for open: $100M\Omega-\infty$. Unfortunately, these ranges of values are too broad and, in practice, extremely time-consuming to handle with mere brute-force simulation.

The time-domain algorithm for fault simulation work in two iterative shells. One is the outer iteration that steps through the time instants such as $t_{n+1} = t_n + h_n$, n=0,1, *etc*. The second iterator steps through the instants t_{n+1} inside the outer iterator to solve the nonlinear equation by applying the Newton-Raphson (NR) method. A nonlinear circuit equation is solved in each iteration of the NR method. A linear system is solved in each NR iteration with Lower–Upper (LU) factorization [48]. In [49], a new method for transient fault simulation for nonlinear analog circuits has been presented. An operational amplifier and an active band-pass filter were selected as test cases. Resistive bridge defects were chosen in this article as structural kinds of defects. In this technique, the speed of fault simulation is measured in terms of simulation latency.

Simulation latency means computational redundancy, e.g., avoiding the repetition of unnecessary specific operations in the simulator. Another approach has been proposed in [50] for fast-time domain simulation. In this work, fast simulation can be carried out by combining different techniques (hierarchical simulation, on-the-fly-decrease of the list of defects, prediction of neighboring problems by implicit sensitivity) together with the enhancement of hierarchical modeling using extra ports.

Fault simulation using high-level models

The fault simulation can be performed by using high-level models. Different techniques are proposed in the literature and exploit behavioral level fault simulation or a hybrid multi-level fault simulation.

Behavioral level analog fault simulation

Fault simulation and injection can be performed at a high level, as described in [51]. Applying behavioral models is one of the methods proposed in the literature to increase fault simulation speed. These behavioral models can be a set of equations relating inputs and outputs or can be several lines of micro-code. Verilog-A, Verilog-AMS, VHDL-A, VHDL-AMS, and System-C are behavioral modeling languages used for accomplishing this task. VHDL-AMS simulators are powerful at a higher level and can make the modeling of analog or mixed signals easier, speeding up the simulation time. Behavioral modeling [52] and simulation of failure modes in analog blocks are possible. However, it also involves a comprehensive analysis of the possible faults that mimic the behavior of a defect. While such behavioral modeling is possible using SPICE macro-models, described with VHDL-AMS or other analog hardware description languages. These techniques are possibly best if used in multi-level fault simulations. The idea of modeling faults at the behavioral level has been discussed in [53, 54, 55, 43, 56, 57, 58, 59, 60, 61, 62, 63] to speed up the fault simulation process.

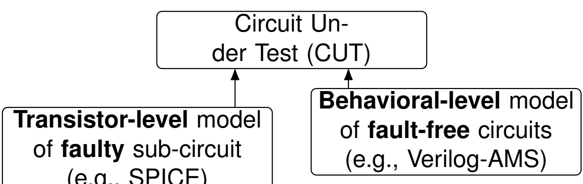


Fig. 3.14: Representation of multi-level modeling.

Multi-level/mixed-mode analog fault simulation

Fault simulation can be speed-up by using multi-level hierarchical analog fault simulation techniques. In multi-level fault simulations, defects can be modeled at the transistor level, whereas fault-free parts of the circuit can be modeled at the behavioral level. In some articles, the multi-level fault simulation is referred to as mixed-mode simulation. Multi-level hierarchical analog fault simulation refers to the use of behavioral models for components defined at the transistor level, defect injection at various abstraction levels, and hierarchical handling of all different definitions of circuit components and defects during the process of fault simulation. Multi-level hierarchical analog fault simulation is a valuable method for dealing with the complexities of analog circuits and producing test signals with high defect coverage [58]. The simulation time can be decreased by using behavioral models for some components of the circuit. Behavioral models have a small number of variables, and the systems of equations representing the circuit's behavior have less

computational complexity [64]. In multi-level fault simulation, the defects can be modeled at the transistor level, while the fault-free parts of the circuit can be modeled at the behavioral level to improve the overall speed of fault simulation [58]. The primary drawback of this technique is that defects in a circuit can take other parts of a circuit out of their normal operating regions. A similar technique is explained in [54], with the difference that defects are modeled at the behavioral level through the Verilog-AMS language while the circuit remains described at the transistor level.

Fault simulation using fault grouping/equivalence

Fault grouping is another direction in which research has been conducted in the past decade to reduce the entire computation time required to perform a defect injection campaign. Grouping of faults can be performed using both transient simulations as well as frequency simulations. Figure 3.15 shows the general flow used in different fault grouping techniques to reduce the number of defects to simulate. The details for the most relevant grouping techniques are shown in the following sections.

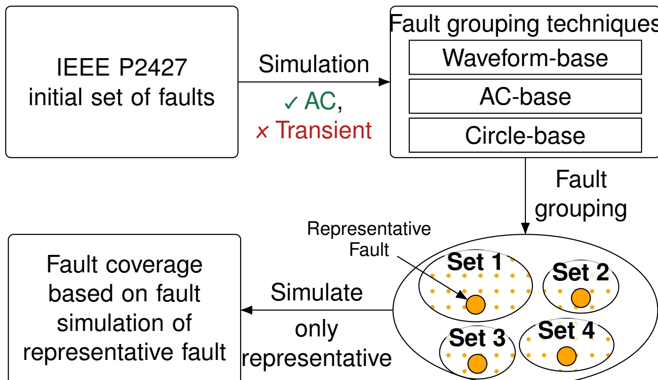


Fig. 3.15: Overview of different fault grouping techniques.

Fault grouping using transient simulation

The primary objective of fault grouping is to subdivide faults into groups [65]. Grouping can be performed by using different strategies related to the simulation method. Various authors have proposed several algorithms for grouping faults [66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77]. The ultimate goal of grouping is to reduce the number of faults that must be simulated by creating clusters of similar/related faults and simulating just one representative fault for each cluster. Fault grouping can be performed in both transients as well as frequency domains. A dynamic fault grouping method to improve the concurrent fault simulation was presented in [66]. This algorithm is also applicable to non-linear circuits. The primary aim of this paper was to reduce the computational cost of fault simulation. In this work, faults were grouped dynamically at every time step. After the grouping, faults presented in every group can be simulated parallel simultaneously. Every group had different time steps, but faults in each group had the same time step. Another approach for fault grouping was introduced in [72]. In this method, hierarchical clustering has been proposed for fault grouping. This method was presented for grouping faults at the component level. One drawback of the proposed work was that by adopting this technique for every fault present in the group, circuits need to be simulated once for every fault, which is time-consuming.

In [73], a fault list compression using the stratified fault grouping technique has been proposed. This approach used stratified fault grouping for the identification of representative faults. This technique can

reduce the number of randomly selected defects needed to achieve a target confidence interval. This method is generic and independent of the fault models used and can be applied to more complex fault models.

Fault grouping using AC simulation

Fault simulation using transient simulations is time-consuming. To overcome the problems of transient analysis in [78, 79, 42], frequency-based analysis techniques have been proposed to reduce the simulation speed of analog defects. The work of [78] was related to grouping the faults in the frequency domain for solving problems of fault diagnosis in large analog circuits. In [79], authors presented a fault clustering technique combining faulty DC Operating Point (OP) and frequency domain analysis. That technique requires N transient simulations (i.e., one for each injected defect), and then, at each OP, they perform an AC analysis.

Concurrent and parallel fault simulation

The fault simulation of analog circuits is more challenging as compared to the fault simulation of digital circuits. Only information associated with a part of the circuit where faults are propagated is utilized in the simulation of digital circuits. While the simulation of analog circuits impacts on the current and voltage levels across all the branches and circuit nodes. Concurrent [80] and parallel fault simulation [81] are considered efficient tools for fault simulation for digital circuits. Nevertheless, fault simulation in analog circuits is often achieved by repeating the process of injecting the defects systematically, requiring to consume a lot of time. As a result, concurrent analog fault simulation techniques have created a new application field in the analog domain [41]. Performing a series of sequential transient simulations is a standard fault simulation method. The different runs are usually not related to the information acquired from the previous runs. A parallel transient simulation method is proposed in [82] to significantly reduce the simulation time by simultaneously simulating many defects with a transient analysis. This technique also provides better performance because the simulation allows the reuse of some results and structures which are obtained from the previous runs [83, 84]. In another paper [80], an efficient fault simulation method has been proposed to simulate defects with a DC and transient simulation simultaneously. The division of the fault groups is dynamic depending on their impact on transient response. The complexity is reduced using novel techniques, including state prediction, RFM computation, and fault order. Outcomes of corresponding fault-free and faulty circuits are shared during the simulation process to speed up the simulation. Consequently, sharing outcomes from one fault help to simplify the next one. A parallel paradigm-based approach has been discussed in [85] for automating fault simulation in analog circuits. The key concept was to use several computational resources for the simulation of defects in parallel. In [86], authors also proposed a parallelization approach to achieve high throughput for a series of transistor-level fault simulations.

Monte Carlo simulation

Monte Carlo [44, 87] is one of the most widely used methods to model parametric variations based on random combinations of values that are selected within the range of each parameter. However, the repetitive random sampling process is required for each defect to acquire more accurate results on the behavior of a circuit. This phenomenon makes the Monte Carlo method very expensive in terms of processing time [88, 89]. Behavioral modeling and inductive fault analysis can be applied to enhance the processing time of the simulation. Monte Carlo simulations require that all the time is consuming. A sampling technique has been proposed to overcome this problem in [90, 91]. This sampling technique is easy to implement with respect

to other techniques. In [44, 92], authors made a reasonable effort to reduce the simulation time of the Monte Carlo technique. Monte Carlo methods usually generate a large number of samples of predictable manufacturing deviations in a circuit and then only simulate those samples that are most likely to generate failing or marginally passing circuits. Hence, most of the time needed to assess the defect parameter value distribution is spent in Monte Carlo simulations. Generally, analog designers spend much time with these types of simulations. In the absence of known distribution of the defect parameter value, a uniform distribution is used to inject the defect. The main shortcomings of this technique are the possibility of generating an unlimited number of parameter variation combinations. But, usually, this is not feasible because nobody wants to increase the number of parametric variations to be simulated.

Inductive fault analysis

Fault simulation speed can be reduced by reducing the number of defects to be simulated by using inductive fault analysis [74], reducing the simulation complexity by behavioral modeling, and reducing the equation set-up time by using the cache mechanism [59]. An efficient flow approach has been proposed in [93], and the method is demonstrated to be valid for a large-scale industrial analog circuit. The primary contribution of this work was the automatic extraction of possible defect locations, computing the likelihood [94, 89] of each defect, and calculating the defect coverage of the circuit test list. The test list can be optimized by analyzing the results while changing the design in order to keep the highest quality level. Selecting defects to simulate based on their probability of occurring can reduce their number but still abide by a given confidence interval of the test coverage estimate. To simulate each faulty circuit, the inductive analysis method employs the NR algorithm. The main idea is to organize the simulation results so that the previous faulty circuit's solution can be used as a good starting point for simulating the next faulty circuit. For all the faulty and the good circuits that have similar behaviors, a one-step NR iteration is performed to create a good ordering.

Fault sensitivity analysis

Fault Sensitivity Analysis (FSA) [45, 95, 96, 97, 98] is a technique for the efficient simulation of analog defects without affecting the simulation accuracy of non-linear circuits while performing transient analysis. This methodology can speed up the simulation process by two orders of magnitude for each defect. In [1], defects are injected only for a specific duration to reduce the overall simulation time as shown in Figure 3.16. Time points are shown in the (a) part of the figure for fault simulation of a priority chosen conductance bridge. A large number of time points are used to execute the standard transient simulation. Meanwhile, the number of time points with test measurements is low, as shown in the (b) part of the figure. The aim is to compute the outcome of the faulty circuit only at the measurement time points.

DC fault simulation

A method for the efficient DC fault simulation of nonlinear analog circuits was presented in [99]. The main focus of this technique was the reduction of the Newton-Raphson iterations to acquire the exact fault simulation using the fault ordering approach. Fault ordering is a way of ordering detectable faults with high precedence to reduce the total simulation time. The idea of using fault ordering in this article was to sort the faults by the closeness of the approximate solutions derived from the one-step relaxation method. The previous faulty response is used as a good starting point for the next faulty circuit. In [100], an approach for fault analysis of the DC domain has been presented that reduces the number of required simulations. In

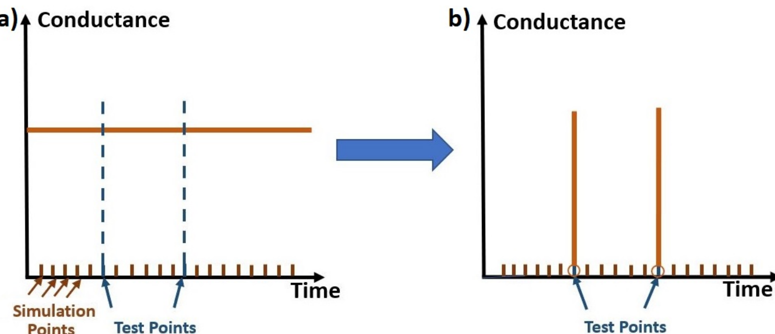


Fig. 3.16: Fault sensitivity analysis overview (see [1]). The left side shows the calculation of the results at every time point, while the right side shows the calculation of results only on selected test points.

the initial step, nonlinear equations are solved during DC simulation. It can be done by solving nonlinear equations using the Newton-Raphson method. The resulting system is solved, which consists of linear equations. The number of simulations required to determine the resistance between arbitrary nodes of a circuit is decreased by using the numerical technique of DC fault analysis proposed in this study. The total simulation time for the selected faults can be reduced by using many CPUs in parallel. In this work, the main focus is on reducing the simulation time of specific defects. One strong point of this method that uses DC simulation is that it is also applicable to non-linear circuits.

Other simulation methods

There are other methods in the literature regarding defect injection and simulation of analog circuits. In [101, 102, 103], authors proposed different methodologies to speed up the simulation process by simulating functional level models. In [101], faulty and fault-free models are abstracted from behavioral (Verilog-AMS language) to functional level (C++ language). Another method proposed in [104] uses graph techniques applied to partition the circuit into independent sub-circuits. The performance of fault simulation on sub-circuits is expected to be more time efficient than simulating the complete circuit at once. A novel approach has been presented in [88] for test vector generation and parametric fault simulation. Statistical models of the faulty circuit and fault-free circuits are generated based on the sensitivity and the process information of the principal components of a circuit. In [105], an effective technique for the simulation of multiple catastrophic defects, either open circuits or short circuits for AC simulation, is proposed. The technique uses the well-known Householder formula from matrix theory to determine node voltage discrepancies caused by changes in certain circuit components. The major contribution of the paper is to provide a systematic method for simulating various combinations of catastrophic defects.

3.2.2 Research/tooling gaps and future directions

This section describes the possible links between those techniques and new approaches in this research field. Defect injection techniques are mainly characterized by two aspects: 1) which defect model they inject and 2) the location where it is injected inside the Device Under Test (DUT). Defect models represent an unwanted physical change in a circuit element or connection that is not within fabrication requirements. Unfortunately, there are several ways of modeling the physical behaviors of a defect, and this has led to the definition of different modeling techniques. Furthermore, each defect model can be injected into several locations of the DUT. Given the number of defects that need to be injected based on the combination of those two dimensions (i.e., defect model and location), simulating all of them is infeasible. A designer must

find a meaningful subset of all possible defects that allow computing fault diagnostic metrics properly and in the shorter time possible. Without common guidelines on how to model defects, choose defect locations, and perform defect selection, each designer is bound to use user- or company-defined techniques and metrics. The first attempt to standardize analog defect models for the different components of SPICE-based languages is the IEEE P2427 draft standard [32], currently under final revision. Some works have been proposed to generalize the analog defect models by defining them at the behavioral level with the Verilog-A language [54]. These behavioral fault models are injected directly into transistor-level descriptions by using pre-processor commands. Consequently, this approach is challenging to apply but can be the first work that tries to define generic defect templates suitable for different simulators.

All the state-of-the-art techniques related to defect injection and fault simulation listed in this section aim to reduce the overall simulation time required to perform a complete defect injection campaign based on the defect universe. These techniques can be subdivided into three groups, those techniques that 1) try to reduce the simulation time required to simulate one defect at a time, 2) try to reduce the total number of defects to be simulated, and 3) combine both the previous techniques. Performing a global defect injection campaign for a DUT has become a time-consuming task due to the ever-increasing size of analog circuits. These techniques are usually bound to (or developed for) a specific simulator, which heavily hinders the efforts of both industry and academia to improve them. The trend is to move away from continuous-time models of computation and explore instead discrete-time or event-driven ones. However, this transition requires generating new abstract models, starting from the original SPICE descriptions. Consequently, many works in literature are related to techniques that try to reduce the overall number of defects to be simulated by relying on the equivalence of the fault behaviors or using abstracted behavioral-level models. Another criticality of some techniques is related to the probability distribution of the defects used to compute the appropriate set of defects to be injected, e.g., likelihood-based techniques. To improve these, techniques are needed to relate the design layout of ICs automatically with these probability distributions associated with the defect to define appropriate metrics for each defect.

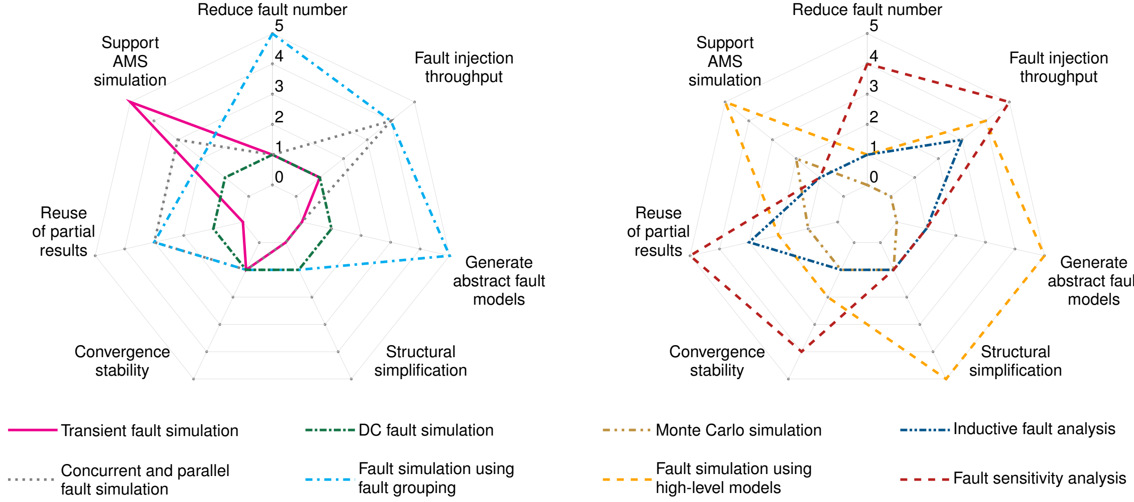


Fig. 3.17: Comparison between the different fault simulation techniques using different criteria: the ability to reduce the number of defects, fault injection throughput, ability to generate abstract defect models, structural simplification, convergence stability, reuse of partial results, and support Analog and Mixed-Signal (AMS) simulation. The score is related to different criteria used to evaluate the different simulation techniques, and it goes from zero to five, where zero means that the corresponding technique does not meet the required capabilities, while five means that the technique can efficiently implement the required capabilities.

Figure 3.17 compares the fault simulation techniques presented in the survey based on different criteria. The criteria used to create the comparison are:

- ability to reduce the number of defects to simulate: the overall number of defects included in the fault injection campaign;
- fault injection throughput per unit time: the time required to perform the fault injection campaign;
- ability to generate abstract defect models: identification of failure modes;
- structural simplification: ability to replace peripheral blocks with abstract representations;
- convergence stability: risk of non-convergence of the simulation;
- reuse of partial results: the ability to reuse previous solutions to increase the solver efficiency;
- AMS simulation support: the ability to support mixed analog-digital simulations.

These metrics make it possible to highlight the key technical characteristics of each simulation technique proposed in the previous sections. Each metric has an associated score ranging from zero to five. A score of zero means that the considered technique does not meet the corresponding capabilities, while a score of five means that the technique can efficiently implement the corresponding capabilities. For example, the techniques that perform fault simulation by exploiting fault grouping have associated a score of 5 in the metric ability to reduce the number of defects to simulate because none of the other techniques have similar technical characteristics. While for example, simulation using fault sensitivity analysis techniques is associated with a score of 1 in the metric AMS simulation support because its technical capabilities to support co-simulation of the analog and digital part is less supported. A possible research direction to reduce the number of defects to simulate is combining layout-level descriptions with transistor-level descriptions to annotate the latter automatically. By exploiting this concept should be possible to inject and simulate only the defects that are reasonably likely contained in the defect universe as specified in the IEEE P2427 draft standard. Also, [106] showed that simulating only the most likely one can still result in too many defects to simulate for an industrial circuit. Another possible research direction is moving to event-driven simulations, that is, simulations based on discrete-time events in the analog domain. Generally, when people speak about reduced models, they think about Real Number Models (RNMs), which are models based on simple resolution functions. A resolution function makes some assumptions about the circuit, and then when a fault violates, it is easy to identify with checkers. The problem is that these techniques do not allow defects on the interfaces to be modeled, and consequently, only a reduced subset of defects can be described. Recently, the technique attracting the most interest exploits SystemVerilog models refined by a novel proposal, defining linear elements and a resolved, bidirectional signal type called EEnet resolution function proposed by Cadence [107].

3.3 Automatic tool for manipulating transistor-level netlist

The parent of transistor-level design languages is the SPICE language [108]. After SPICE, many other proprietary design languages were developed to describe transistor-level designs [109]. Each vendor has defined its variant of the SPICE language by extending it with the support of new functions and components, trying to satisfy the designer's requests [110]. Nonetheless, starting from any transistor-level descriptions, it is possible to synthesize an electrical circuit printed on silicon. The extension of the SPICE language has been carried out by defining more electrical components (e.g., a MOSFET) that are aimed at modeling complex physical behaviors. Many of these design languages have been created by vendors who have created and marketed tools to manipulate them. These tools provide a programmatic way to manipulate these languages, but they are complex and difficult to use outside vendor-designed development environments.

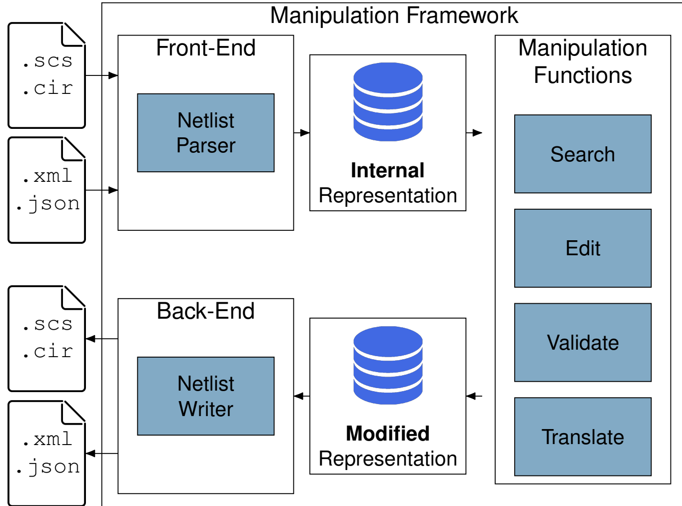


Fig. 3.18: Overview of the proposed manipulation framework.

The idea underlying the entire work is to allow designers to manipulate transistor-level descriptions with the proposed framework without using proprietary software [111]. No new support is given to simulate these transistor-level descriptions based on analog simulators that reflect state of art for analog simulation, e.g., Eldo and Spectre. Instead, a new framework is proposed to manipulate transistor-level descriptions expressed in different languages, creating new descriptions that analog simulators will then simulate. The manipulation framework structure is presented in Figure 3.18, which comprises front-ends and back-ends. This common framework is based on the fact that all the transistor-level languages that exist on the market have a different *syntax*, but usually a common *semantics*. The same concept is valid for the comparison between two other behavioral transistor-level languages, VHDL-AMS and Verilog-AMS [112]. Moreover, the proposed framework is coded with the C++ language and supports full binding with the Python language by exploiting the Pybind11 library. [1] The main innovations introduced with the presented framework are:

- Definition of SPICE, Eldo and Spectre grammar in ANTLR v4 [2];
- Creation of a manipulation framework with a common internal structure that exploits the common semantic between the transistor-level languages;
- Definition of the front-end and back-end for Eldo, Spectre, XML, and JSON;
- Application of the framework to real use-cases *e.g.*, wrapping of subcircuits, injection of defect models, graph visualization of the model structure.

3.3.1 State of the art and definitions

Circuit simulation is an effective method for manufacturers to simulate analog circuits before fabrication. Newly complex analog electronic circuit designs can be rapidly evaluated and validated on circuit simulators. SPICE, Eldo, and Spectre are the simulators used to simulate transistor-level descriptions of analog circuits. UC Berkeley University developed SPICE [113] language that resulted in the development of several versions such as Spice2, Spice3, PSpice [114], and HSpice [115]. SPICE has a long history of releases, from 1975 to 1993. PSpice and HSpice were developed by Cadence and Avati/Meta Software in 2001, respectively. Spectre is another SPICE-class circuit simulator tool with its associated language [116], created by Cadence. It supports the fundamental SPICE analyses: DC, AC, TRAN, and HB. Moreover, it supports

[1] https://github.com/pybind/pybind11
[2] https://www.antlr.org/

interfacing between transistor-level netlists and Verilog-A and Verilog-AMS descriptions [117]. There is also an enhanced version of Spectre that provides support for RF and mixed-signal simulation. Spectre and Eldo are considered leading circuit simulators. Both offer a simulation of netlists in many ways, e.g., frequency and transients analysis. It also has high performance in processing speed as compared to SPICE-based simulators. Spectre and Eldo have different syntaxes with a common semantic. Due to their complex syntax, there are no publicly available frameworks that offer a unified methodology for their parsing, writing, and manipulation.

Code manipulation strategies

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

The tokens recognize through the lexer acquire an individual semantic value. It does not consider their semantic value in the context of the whole program because this is the duty of the parser. The parser organizes the tokens into allowed sequences of the language's grammar. If the language is used as defined in the grammar, the parser will recognize patterns that constitute specific structures and group them appropriately. If the parser encounters a sequence of tokens that matches none of the allowed sequences, it will raise an error and perhaps try to recover it by making assumptions about the nature of that error. The parser checks that the token conforms to the language's syntax defined by the grammar. Usually, the parser converts the sequences of tokens for which it was built by matching them to another form, such as an AST. An AST is easier to translate into an objective language because it implicitly contains additional information due to its structure and is an essential part of the language-translation process. When for a language, the lexer and parser are defined, ANTLR allows the definition of rules that the lexer should follow to tokenize a stream of characters and rules that the parser should use to interpret the stream of tokens. ANTLR can generate a lexer and a parser that can be used to interpret programs written in one language and translate them into other languages and ASTs. The ANTLR project allows many extensions and has many applications.

State of the art: manipulation tools for transistor-level languages

Various simulators are available for the simulation of the transistor level circuits [116, 118, 109]. Although these simulators have different language formats or syntax, they follow the same standard semantics. In addition, researchers are attempting to design tools to convert and manipulate languages from one to another. Original SPICE language resulted in several transistor-level languages. However, limited tools are available for language conversion and manipulation.

These tools can convert libraries and circuits between these transistor-level languages, e.g., Spectre to Eldo converter (spect2el), Netlist Translator for SPICE, and Spectre[119]. Spectre to Eldo converter is available in the Eldo simulator that allows converting a Spectre circuit to the equivalent circuit described in Eldo [118]. This tool supports AC, DC, Transient, S-parameters, Monte Carlo, Sweep, Noise, Sensitivity, and Transfer function analysis. There are several limitations with Spect2el as it is a uni-directional that converts only from Spectre to Eldo. It cannot convert the circuit properly if any section is written in SPICE spect2el, and in some cases, this requires manual efforts for modifications. *e.g.*; Netlists and libraries should use Spectre syntax. Any part of or section of a program written in SPICE syntax cannot be correctly converted into the desired format. Furthermore, netlists and libraries cannot be handled in case-sensitive. Both Eldo and Spectre languages have some similarities but differ in many aspects. Currently, there is no available framework that provides unified semantics. Above mentioned Netlist translator [119] was developed to simulate the Spectre and SPICE (Spice2, Spice3, PSpice, and HSpice) in Advance Design System (ADS). In [120], a Python interface for SPICE-based simulations were presented. That interface aims to help the designer in the industry solve circuit sizing problems of integrated analog CMOS circuits for SPICE-based simulations.

3.3.2 Language independent framework

This section presents the language-independent framework that allows manipulating transistor-level descriptions easily. The entire framework is open-source and available on GitHub. [3] In figure 3.18, a schematic version of the manipulation framework is presented. Several front-ends bring heterogeneous descriptions into an in-memory description based on the AST generated by the parsing phase within the framework. These language descriptions that the framework reads as input can be written in Eldo, Spectre, or XML languages as output. A grammar for Eldo (the Eldo grammar also covers SPICE) and Spectre written in a format compatible with ANTLR are provided with the framework. After the parsing phase, the structure of the considered AST will be fully represented within the internal structure of the framework. This means that no information will be lost during this transcription, from the input description to the internal structure of the framework. If required, it is possible to specify through specific settings whether to skip the parsing of the comments or not. In the following sections, the entire internal structure of the framework will be described in detail. The tool allows for performing several manipulation operations on the internal structure, such as changing the master of a component or the subcircuit pins. Useful real applications of the framework are described in Section 3.3.4. After applying manipulation techniques, the internal structure of the framework is printed through several back-ends. It is possible to print in the output the internal structure as Eldo or Spectre netlist or to XML or JSON files. When printed in output as Eldo or Spectre code, the manipulated descriptions are ready to be simulated with commercial simulators. Furthermore, this framework can convert from Eldo to Spectre and vice-versa. The conversion between different transistor-level languages is based on the equivalence of the semantics used to describe transistor-level components. The framework's support to read and write JSON descriptions allows the designer to build pipelines of tools built upon the proposed framework's APIs. These intermediate tools could be shared among designers and enable re-utilization.

[3] https://github.com/sydelity-net/EDACurry

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

Listing 3.1: Eldo grammar fragment written in ANTLR.

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

Listing 3.2: Spectre grammar fragment written in ANTLR.

Definition of a specific parser for each language

The input netlist should be read in input from the proposed manipulation framework to store all the information inside the internal structure. To read a netlist in input, it is necessary to have a unique front-end for each language that will parse the information read in input and map it to the internal structure of the framework. For this reason, we have developed some grammar following the ANTLR syntax to read input netlists. In particular, we defined grammar for the Eldo language and one for Spectre. The Eldo grammar can also completely cover the original SPICE language from which it derives. To define these two grammars, the commonalities relative to the semantics of these languages are exploited. For Eldo and Spectre, a lexer and a parser are defined. Into the lexer, all the possible string tokens are defined. While in the parser, it is defined the semantic of a series of contiguous tokens. Then, through the Antlr library, the empty skeleton of the C++ parsers is generated. The skeleton of the parser generated through Antlr can be a Visitor or a Listener, both top-down recursive-descent. The *Visitor* and *Listener* patterns allow us to visit the Abstract Syntax Tree (AST) generated from the grammar, but there are important differences between them. A *Visitor* pattern allows visiting each element of an AST with specific visit functions. While the *Listener* pattern allows visiting an AST whit an access and exit function for each element of the tree. In the proposed framework, the *Visitor* pattern is used to create the front ends.

Let us focus on the two languages, Eldo and Spectre. The former has a well-defined syntax without ambiguity, while the latter has a syntax that is sometimes ambiguous [118, 116]; for that reason, the complexity of the two proposed grammars is different. Nevertheless, both languages are equivalently used to represent transistor-level netlists. We studied the commonalities between the Eldo and Spectre grammar and defined their parsers accordingly. Consequently, it makes it easier to map the parsed descriptions (*e.g.*, Eldo, Spectre, SPICE) to the internal representation of the framework. In Listing 3.1, a fragment of the Eldo grammar we defined is proposed. While in Listing 3.2, a fragment of the Spectre grammar is proposed. Analyzing these two grammar fragments makes it easy to identify similar categories of elements in a netlist written in Eldo and in Spectre. For example, in common, it is present: subcircuits, control commands, analysis statements, and components in both languages.

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

Listing 3.3: Fragment of the Eldo grammar for the component X.

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

Listing 3.4: Fragment of the Spectre grammar for the instantiation of a component.

Similarities between the Eldo and Spectre languages

Eldo and Spectre syntactically are very different from each other, but the points of contact of semantics are many, as evidenced in the structure of the two grammars shown in Listing 3.1 and Listing 3.2. For example, a subcircuit in both languages may contain definitions and instantiations of components. The parts that can be described equivalently in either language are:

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

Now let us consider the component used in both languages to instantiate subcircuits. The Listing 3.3 shows how you can instantiate a subcircuit with the Eldo language; this is possible with the `X` component. This `X` component is only used to instantiate custom models and not standard models of the language. A standard component in Eldo is defined through components that start with alphabetic letters.

Let us consider the same syntax used for instantiating subcircuits in the Spectre language. The Listing 3.4 shows the generic definition of a component for the SPECTRE language. Different from what happens in the Eldo language, where we have a specific component `X` to instantiate a generic subcircuit, in Spectre, we do not have a specific component. A component, either library or custom (thus, defined via a new subcircuit) in Spectre, is defined in the same way: the component identifier, the list of nodes, the master, and its attributes. Therefore, the master can be a generic subcircuit or a standard library component.

Some language statements cannot be mapped to both languages by analyzing the semantics of all possible statements accepted by these languages. For example, it is impossible to define the Eldo command `.MEAS` in the Spectre language. The problem arises because even at the semantical level, there is no `.MEAS` counterpart in the Spectre language. When the framework tries to generate an equivalent description of the `.MEAS` command for Spectre returns an error. Because it is impossible to map two statements without a semantic shared between the two languages considered. Furthermore, these problematic statements will be omitted from the generated description.

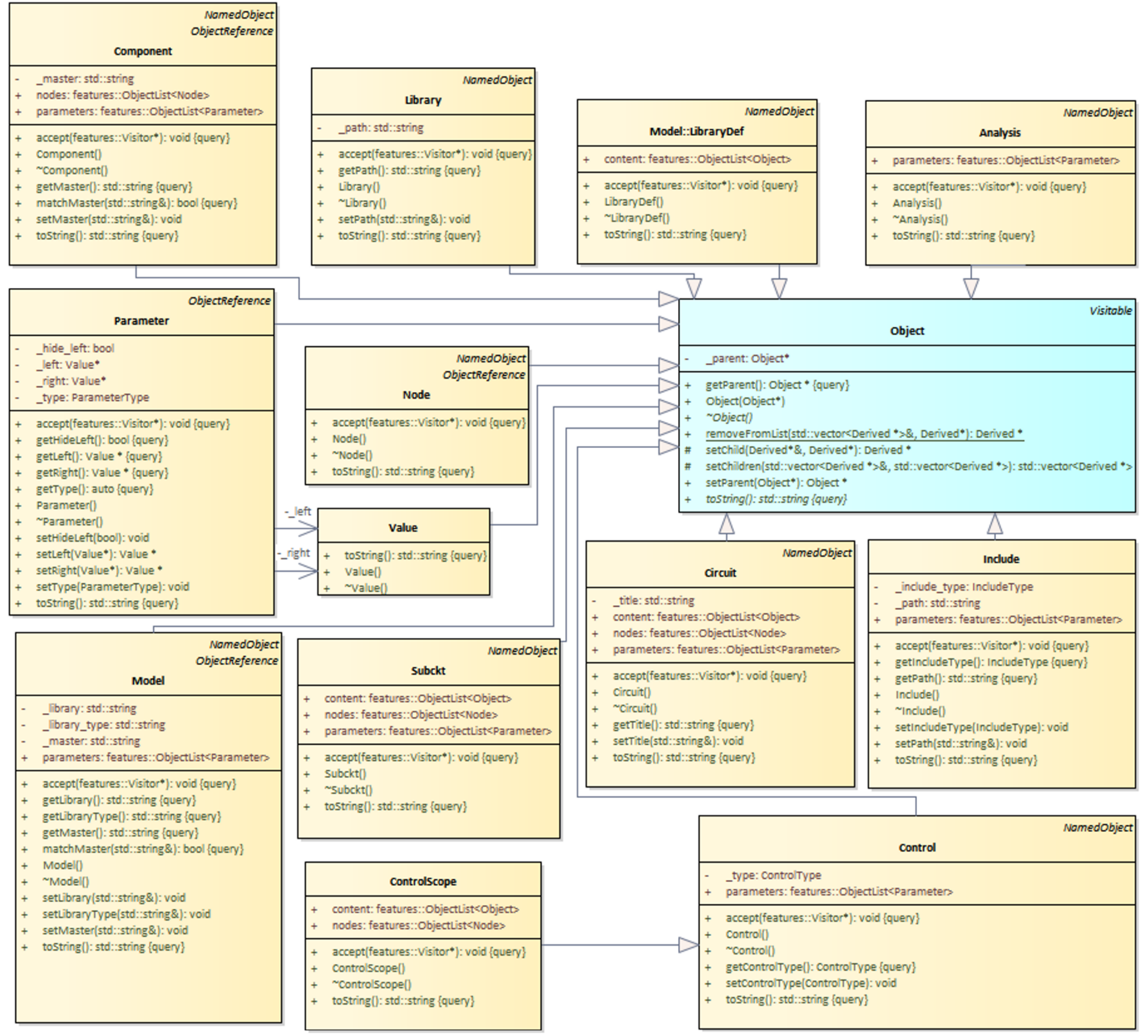


Fig. 3.19: Unified Modeling Language (UML) Class diagram of the internal structure of the framework - part 1.

Framework internal structure

The internal classes representation of our C++ framework is depicted in Figure 3.19 and 3.20 as UML diagram. Each class of the UML diagram represents a fundamental block of a transistor-level netlist. The grammar written for Eldo and Spectre is structured to reflect the internal class structure of our framework. With structured grammar, it is easier to build front-ends to populate the internal representation of the framework. The framework was developed with a unique internal structure that can collect all the netlist information. The concept behind the idea of a unique internal structure for manipulating languages is the basis of the HIFSuite [109] framework. HIFSuite allows manipulating another class of languages: Hardware Description Languages (HDL), *e.g.*, VHDL, Verilog, Verilog-AMS. Usually, a transistor-level netlist contains a circuit (`Circuit` class) that could contain many subcircuits (`Subckt` class) and simulation control commands (`Command` class). Then, inside a subcircuit are stored the components (`Component` class), and for each component, many parameters (`Parameter` class) and assignments to a parameter (`ParameterAssign` class). Moreover, a subcircuit had defined its pins (`Node` class) to interconnect it with other designs. In detail, the UML diagram in Figure 3.19 presents the set of classes necessary to save all the netlist elements, *e.g.*, `Circuit`, `Subckt`, `Model`, `Control`, `Include`, `Node`, `Parameter`, `Component`, `Library`, `Analysis`.

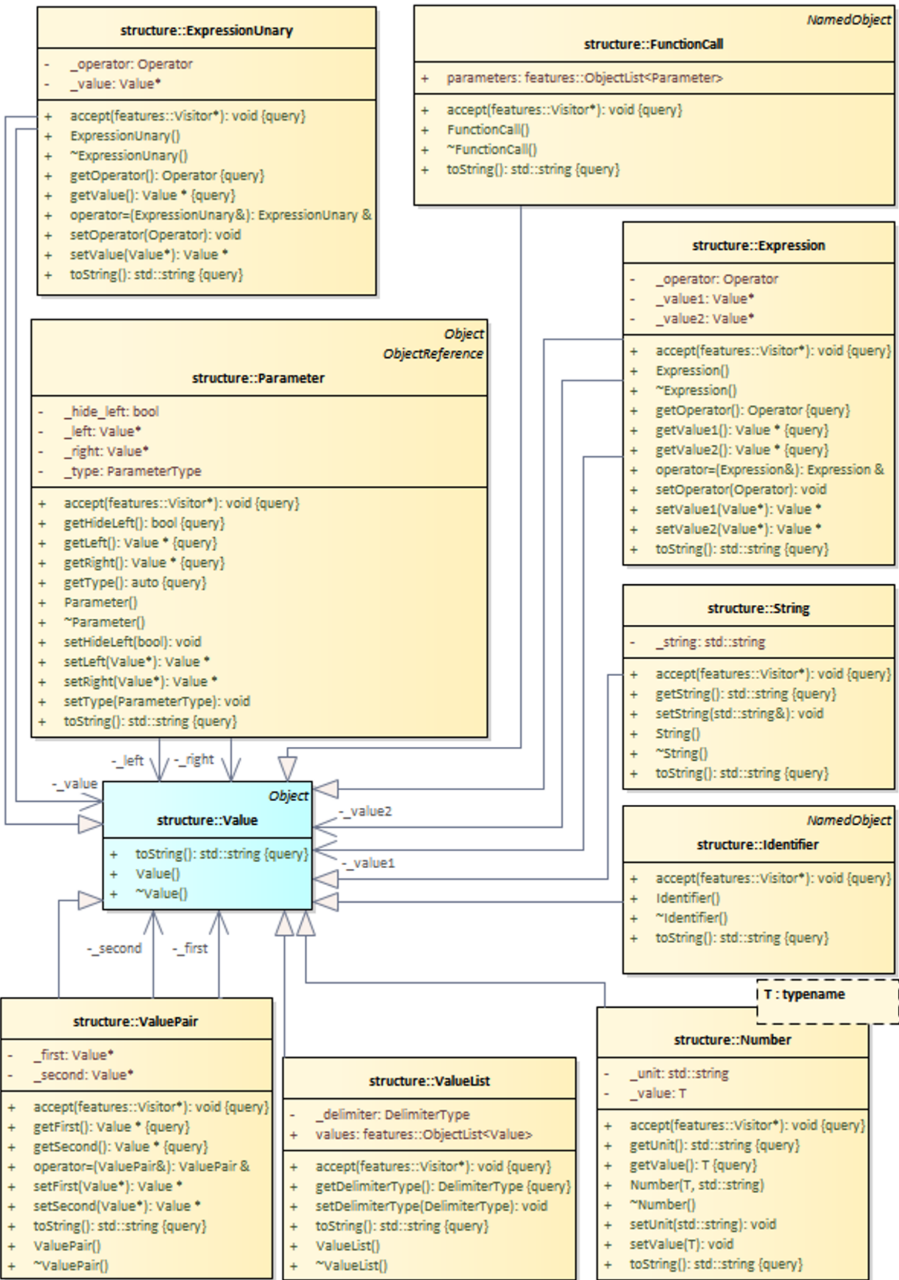


Fig. 3.20: UML Class diagram of the internal structure of the framework - part 2.

All these classes that represent netlist elements implement the functions of the abstract class `Object` (non-dashed arrow with a full point towards the extended class). It is easy to build an internal tree-structured representation of all the netlist information with this class structure. This proposed structure of the framework implements the *Composite* design pattern, necessary when it is required to compose objects into tree structures. This is because the class `Circuit`, for example, has a set of ports (represented as a list of `Node`), a list of parameters (represented as a list of `Parameter`), and a content (represented by a list of `Object`). In this way, a `Circuit` class can structurally contain other elements of a netlist that implement the `Object` class. In addition, the *Visitor* behavioral design pattern is implemented by adding the `accept()` function to each class. In this way, it is simple to visit each element of the internal struc-

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

Listing 3.5: OPAMP subckt described in Eldo language.

ture through a visitor or listener class to create a dedicated back-end to print in the output the structure information.

The second UML diagram in Figure 3.20 presents the set of classes necessary to save the netlist values. A value (`Value` class) could be a number (template's class `Number`), a string (`String` class), an expression (`Expression` class), *etc*. The `Value` class is defined as abstract. Consequently, the derived classes should implement their interfaces. Moreover, all these classes that represent netlist values have an `accept()` function necessary to visit it with a visitor. Other design patterns besides the *Visitor* design pattern are used within the proposed framework, for example, the *factory method* and *composite* design patterns. Through the design pattern *factory method*, part of the creational patterns, it is easy to instantiate objects that are part of our internal structure. Similarly, the front-end created from a specific grammar will use the functions provided by the `Factory` class to instantiate objects by avoiding instantiation issues. Another design pattern used for the debug print is named *Template method*, and it is a behavioral design pattern. Following this design pattern, the skeleton of an algorithm is defined into the abstract (or superclass), and into the subclasses, the same method is overridden by adding additional steps to the algorithm without changing its structure.

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

3.3.3 Framework validation

The framework presented in Section 3.4.3 has been validated with different designs with open access. In particular, to show how it is possible to define the same design with different languages by keeping the same semantics, it is considered the model of an Operational Amplifier (OPAMP), presented in Figure 3.1. This model is retrieved from the analog benchmarks suite available online [35].

The Listing 3.5 shows the subcircuit for the model of the OPAMP described using the Eldo language. This electrical network is composed of several components, especially MOSFET, which are interconnected to define the behavior of the OPAMP. The OPAMP model is an integrated circuit that can amplify weak electric signals. The component parameters are exemplified with the token `<parameters>`. The Listing 3.6 shows the subcircuit for the model of the OPAMP described using the Spectre language. This electrical network is precisely the same as specified in the Listing 3.5 but written in Spectre. Comparing these two

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

Listing 3.6: OPAMP subckt described in Spectre language.

listings described in Eldo and Spectre, we can see that the syntax is different. For example, to specify the list of pins of a component in Spectre, we use the brackets, while in Eldo, these are missing. However, it is possible to model the same physical behaviors for a given design starting from a different syntax and to know the two languages.

AST generation through grammars

Starting from the grammars defined for Eldo and Spectre (lexer and parser) developed per the ANTLR library, we show how the AST generated by ANTLR from our grammars are different for the same component described using the Eldo and Spectre languages. The proposed framework exploits the capabilities of ANTLR and our grammars to be able to read as input any description written in either of these languages. Through a Visitor that reads every portion of the AST that is generated by ANTLR during the parsing phase, we can populate our internal representation of the framework, which was developed in order to store all the information of both Eldo and Spectre designs, without losing any information. In particular in Figure 3.21 is reported the AST of the first component of the `OPAMP` subcircuit (see Listing 3.5), while in Figure 3.22 is reported the AST of the first component of the OPAMP subcircuit (see Listing 3.6). From these two ASTs is clearly visible that the differences between the two trees are many. Nevertheless, these syntax differences do not impact our transistor-level model's semantics (behavior) when we describe it in a different language.

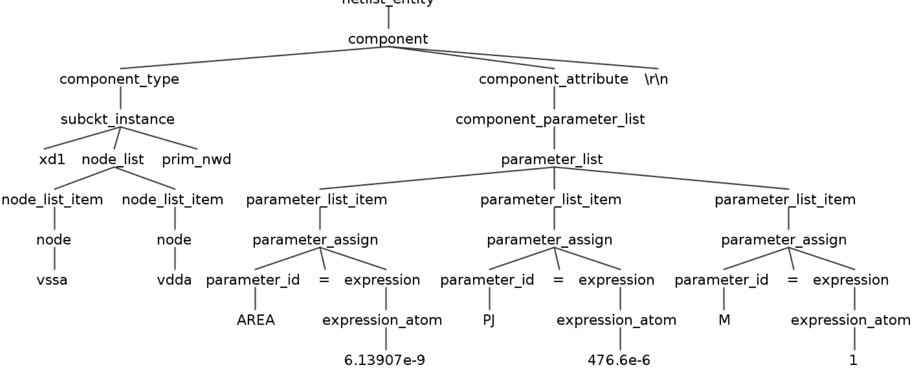


Fig. 3.21: Abstract syntax tree of the OPAMP `xd1` component described in Eldo language.

Semantic equivalence validation through simulation

To show an equivalent design in both Eldo and Spectre languages by exploiting the semantics commonalities, we defined the same design, the OPAMP taken from the analog benchmarks suite [35] originally defined in SPICE and adapted in Eldo and Spectre. The transient simulation is used to validate the behavior of both designs and verify the equivalence. Figure 3.23 shows the results of transient simulations of the same design, the OPAMP 3.1 described in both Eldo and Spectre. The blue line (marker x) is used to simulate the design described in the Eldo language and simulated with Eldo. The purple line (marker o) is for the simulation of the same design described with the Spectre language and simulated with Spectre. By analyzing this graph, it is possible to understand how the behaviors of the two simulations of the OPAMP are precisely equivalent. The plotted waveform has a difference of less than 10% error.

3.3.4 Framework applications

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

The proposed framework can efficiently perform several tasks:

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

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

Subcircuit wrapping

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

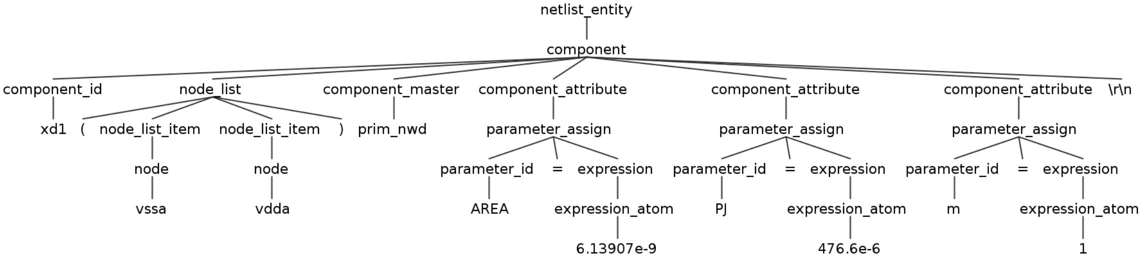


Fig. 3.22: Abstract syntax tree of the OPAMP `xd1` component described in Spectre language.

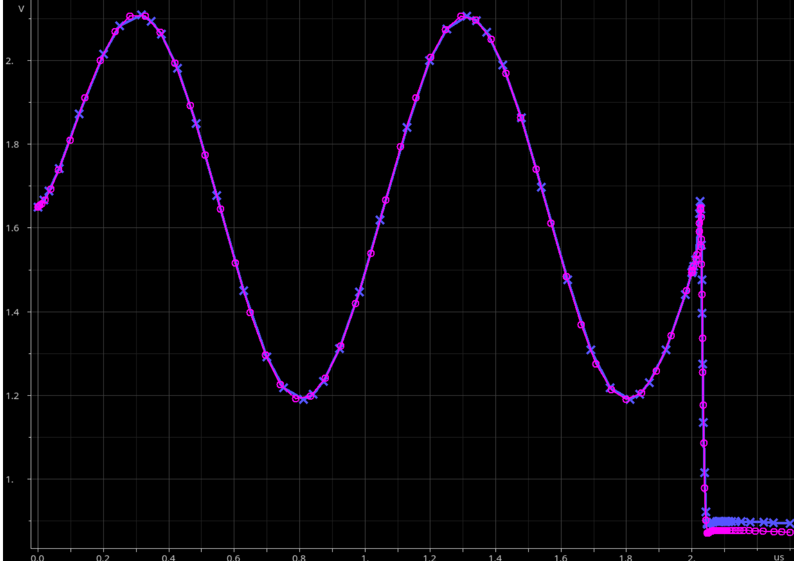


Fig. 3.23: Comparison of the VOUT behavior for the transient simulation of the OPAMP described both in Eldo (identified by a blue line with crosses) and Spectre (identified by a purple line with circles).

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

Listing 3.7: OPAMP subckt wrapped described in Eldo language.

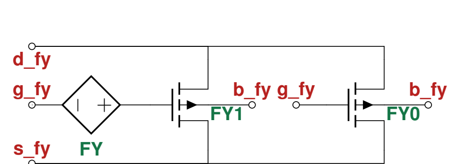


Fig. 3.24: Proposed MOS stuck-on defect model.

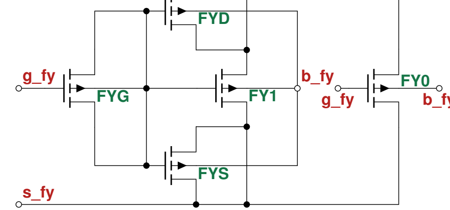


Fig. 3.25: Proposed MOS stuck-off defect model.

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

For example, it is possible to wrap with the internal mode the subcircuit proposed in Listing 3.5. A possible wrapping for the `OPAMP` subcircuit is shown in Listing 3.7. All the internal nodes of the subcircuit became named dut_<node-name>. While the pins of the wrapped subcircuit will be connected to the external subcircuit with additional components. In the example, with another subcircuit instantiated with the component of Eldo `X` necessary to instantiate subcircuits.

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

Listing 3.8: XML fragment of the OPAMP subcircuit generated through our framework.

Injection of defect models

Through the framework, it is possible to inject defect models in a systematic way. Currently, no techniques are implemented to decide which defect models to inject and where based on the failure probabilities (see the work of Sunter [121] for detailed information regarding random defect sampling). The issue of defect models and how to inject them within transistor-level descriptions are still a hot topic of research. The IEEE P2427 Standard [32] is working to standardize the set of possible defects within transistor-level netlists. If we consider the stuck-on and stuck-off defects proposed in Figure 3.24 and Figure 3.25, respectively, it is easy to see these defect patterns as additional subcircuits that need to be added at precise points in our netlist. Usually, these are injected into Eldo via `.ALTER` statement in order to be individually activated during simulations. It is critical to be able to activate one defect at a time and proceed with the simulation to calculate the correct coverage metrics. The procedure followed to inject these defect modes into transistor-level descriptions is presented in Section 3.4.

Print a netlist as XML/JSON file

Another feature of the framework is to output the netlist read as input (and modified internally if necessary) as an XML or JSON description. This is possible by exploiting the two purpose-built back-ends. These two formats allow to preserve the tree structure in which the information is saved within the internal representation of the framework. An example of XML description for the OPAMP subcircuit represented as schematic in Figure 3.1 is depicted in the Listing 3.8. This structure of the XML description is agnostic to the language and represents exactly the internal representation of our framework. This XML could contain all the information of the SPICE, Eldo, and Spectre languages.

Visualize a netlist as a tree

Visualizing the structure and content of a netlist graphically is a feature that can help transistor-level designers to verify if the design created follows the initial constraints. By exploiting the JSON back-end provided

```
1 .param rvalue=3.3k
2 .param length=0.053u width=0.047u
3
4 c5 2 0 100p
5 r5 n3 n4 value=rvalue
6
7 .subckt nw n1 n2 l=length w=width
8     .param rsh=1100
9
10    .subckt rngps n3 n4 l2=l*2 w2=w*2
11        .param rsh2=5.0
12        r2 n3 n4 'rsh2*(l2/w2)'
13    .ends rngps
14
15    r1 n1 1 'rsh*(l/w)'
16
17    x1 1 n2 FOO A=1 B=1 M=2
18
19 .ends nw
20
21 .alter a1
22    r5 1 2 10k
23    c5 2 0 100p
24    .ac dec 15 200 1500meg
25 .end
26
27 .tran 2e-6 2e-3
```

Listing 3.9: Simple netlist modeled with the ELDO language.

inside the framework and combining it with the Graphviz library (based on the DOT language), it is possible to visualize the netlist graphically. [4] Let us consider the netlist proposed in Listing 3.9. This simple example of a netlist described with the Eldo language models the main components that we can find in a netlist, *e.g.*, subcircuit, parameter, and analysis statements.

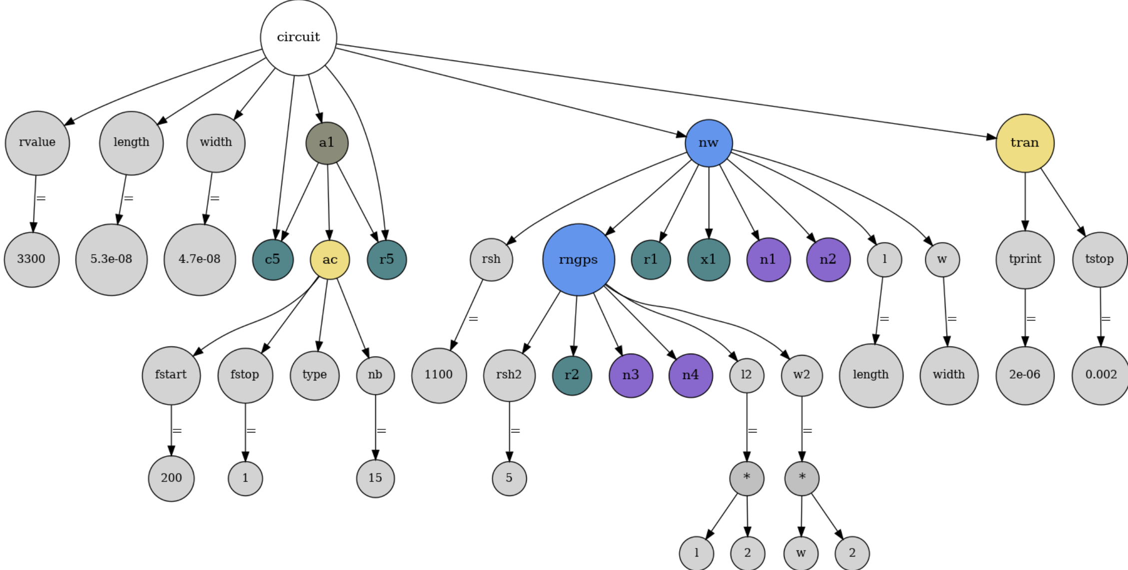


Fig. 3.26: Graphical visualization of the netlist presented in Listing 3.9 produced through the proposed framework.

By reading in input this Eldo netlist with the framework and using the back-end JSON combined with a Python script to convert it into DOT language, it is possible to visualize the netlist graphically. Figure 3.26 shows a possible graphical representation of the netlist presented in Listing 3.9. With a graphical representation, it is easy to understand the structure of the netlist; in this specific example, we have only one circuit (the root of our netlist modeled with a circle with white background), global parameters modeled with circles with a gray background, two subcircuits modeled with circles with a blue background, components

[4] https://graphviz.org/

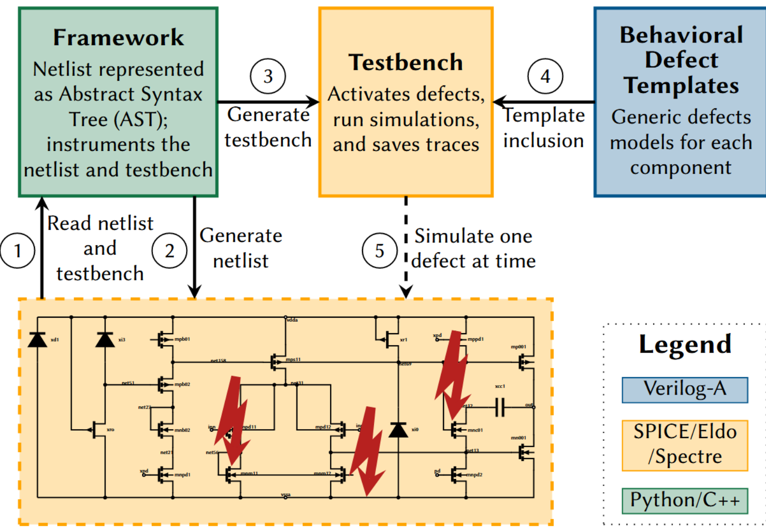


Fig. 3.27: Methodology for describing generic defect templates and injecting them into a transistor-level netlist.

modeled with circles with a green background, ports modeled with circles with a purple background and infinite commands to manage the simulations modeled with circles with a yellow background.

3.4 Verilog-A implementation of generic defect templates for analog fault injection

In the electronic design and testing field, *functional safety* is crucial in automotive and in different domains with mixed-signal applications [8]. The effective fault modeling solutions for analog circuits are lagging compared to digital circuits [9]. In the analog field, there needs to be more standardization of defect models and coverage calculation approaches. The IEEE P2427 working group has laid the foundations for a unified approach to calculate the coverage based on a standard set of analog defects [32]. In analog circuits, defect models are usually injected directly at the transistor level by exploiting different techniques related to commercial EDA tools used to instrument the netlist and then simulate it [122]. These defect models are defined with different templates, usually complex and not portable between the different EDA toolsets. Moreover, implementing an efficient and automatic fault injection campaign with these tools requires experience in this field [89].

This work proposes to provide open and easily modifiable defect templates described with the Verilog-A language. The Verilog-A language allows describing different defect models for each netlist component. Then, based on the defect template library, a simple testbench is generated to perform the fault injection campaign directly at the transistor level. Figure 3.27 presents the overview of the proposed framework. Initially, a netlist of an analog circuit described with a SPICE-based language is parsed to extract all the netlist information. Then, to automatically inject the defect models into a transistor-level netlist, the testbench is automatically adapted to enable a systematic fault injection campaign. This testbench automatically selects the defect locations from a pre-defined list and connects the defect inside the netlist based on the defect templates by activating one defect at a time. After this injection process, all the defects are simulated with SPICE-based simulators to calculate standard coverage metrics for the DUT.

The proposal is innovative because there is no need to create different subcircuits for each defect template that could affect the components of a DUT. In detail, the main contributions of the proposed work are:

- Analog defect templates written in Verilog-A following the IEEE P2427 defect models guidelines;
- Defect templates suitable for all the simulators that support the Verilog-A language without the need to define different defect templates for each simulator;
- Defect templates independent from the underlying model, only the parameter of the model needs to know;
- An automatic tool-chain for the injection of the defect templates that exploits the *netlist-parsing framework*.

3.4.1 State of the art and definitions

This section is further divided into three subsections. The first section explains transistor-level defect models. The second section presented the capabilities of the Verilog-A language. Then, the last section will discuss state-of-the-art related to analog defect modeling and injection.

Transistor-level defect models

For transistor-level models, there is no industry-accepted definition of analog defects. According to the definition provided by the IEEE P2427 draft standard [32], a defect is considered an unexpected change in the physical structure of the circuit. The difference inside the physical structure affects the function of a subsystem that can produce faulty behaviors. For example, changes in the physical structure can be considered excess of gate-oxide charges or excess of interface trap in a Metal-Oxide-Semiconductor (MOS) as described in [36]. Usually, soft and hard defects are the two types of defects considered in analog circuits. The IEEE P2427 draft standard presents differences between hard and soft defects. The random and systemic process variations in analog circuits cause changes in parametric values such as capacitance and resistance, and these parametric defects are usually considered soft defects. The hard defects generate open and short circuits between the connections of components that change the topology of a circuit. The most used components that compose an analog circuit affected by defects are the MOS that can be subdivided into MOS type -n and type -p (schematic description shown in Figure 3.28).

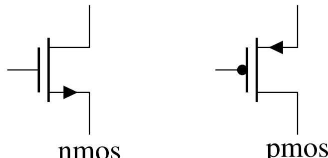


Fig. 3.28: MOS type -n and type -p.

Figure 3.29 presents four different defects that could happen in an N-channel MOS: (a) drain open, (b) source open, (c) gate-drain short, and (d) gate-source short. Open defects are represented with a red X, while short with a red connection between the two MOS pins affected by the defect. The behavior of these defects depends on their impact on the primary current path between the cathode (drain) and anode (source) in MOS transistor by defining three different groups:

- Hard defects can create a permanent direct or indirect conductive path between the cathode and anode. The direct path defect is referred as drain-source short, and the indirect path defect is referred as drain-gate short;

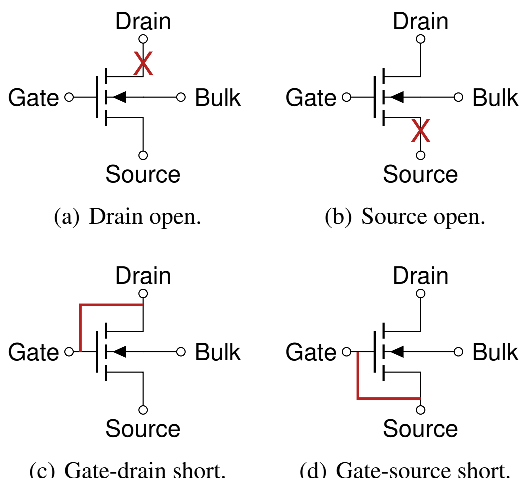


Fig. 3.29: Some examples of faults in a N-Channel MOSFET.

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

Verilog-A language

The design of integrated circuits is complex, and it is challenging for engineers to characterize complex behaviors. HDL, such as Verilog and VHDL, provide various levels of abstraction to design IC. The system behavior can be described at a high level, then gate-level descriptions can be generated using simulation synthesis programs. The interaction between digital and analog signals can be managed with mixed-signal languages. Verilog-A is a subset of Verilog-AMS [112], and it describes the behavior of analog signals with the additional functionality of interfacing some of the behavior of digital signals. Analog behaviors of the conservative systems can be described at a high level with Verilog-A, which supports conditional statements, usually part only of the programming languages. Verilog-A is supported by many EDA vendors, so different descriptions written with this language can be simulated with different commercial tools. By exploiting the potentialities of Verilog-A, new defect models can be developed rapidly at the behavioral level and can be simulated into SPICE designs, as proposed in this article.

State-of-the-art of defect modeling and injection techniques

In the literature, many defect modeling and injection techniques have been proposed for effective fault simulation campaigns. A mixed signal defect injection technique has been proposed in [54], in which the Verilog-A language is used to develop defect models directly onto the circuit devices. The authors present an approach based on pre-processor commands that involve mixed-signal co-simulation of Verilog/VHDL with SPICE. These pre-processor commands can only be set as arguments to the call to the simulator and not by using the simulator sweep command or instance parameters as proposed in our work. In [123], behavioral defect modeling of CMOS circuits has been proposed for fault simulation. In the above-mentioned work, some parts of the circuits and defects are modeled using the VHDL-AMS language. In [124], authors selected a wide range of short and open defects with different resistance values. Open and short defects

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

Listing 3.10: Templates for MOSFET defect models.

have been investigated by injecting the range of $1k$, $2K$, $5K$, $10K$, $100K$, ∞ Ω, and 1, 100, $1K$, $2k$, $5K$ Ω. A new open-gate model for DC simulations has been proposed in [125] after analyzing the open defects in analog circuits. This proposed model was based on voltage-dependent voltage sources, which keep the transistor in the sub-threshold region, as concluded from the in-depth analysis. In [122], authors presented an overview of different defect model techniques present in the literature. According to this article, there is no industry-accepted technique for efficient defect modeling and injection in analog circuits. The impact of various defects, including open, short, extreme variation, or user-defined defects, can be analyzed in a defect simulator called Tessent DefectSim [5]. This simulator works with Mentor Graphics Eldo and Questa

[5] https://eda.sw.siemens.com/en-US/ic/tessent/test/defectsim/

ADMS simulators [109]. Tessent DefectSim first uses a random selection technique to inject the defects into circuits, then efficiently simulates and analyzes the results across multiple testbenches. The main limitations of these approaches are related to the fact that they are not portable between different simulators. Starting from this main problem, we have been constructing the defect templates described in Section 3.4.2.

3.4.2 Verilog-A based fault templates

This section aims to describe how the fault templates represented using the Verilog-A language were constructed; it also compares them to a set of fault templates used in state of the art by the Defectsim tool provided by Siemens EDA. The proposal is to use Verilog-A template descriptions as a portable format among different simulators.

Verilog-A defect templates

The proposed defect templates are based on defect models described by the IEEE P2427 draft standard [32], especially defects that can affect p-type and n-type MOSs will be considered. These defects are described as hard defects and can be related to manufacturing errors of an IC or to a degradation of the physical properties of a circuit due to external causes, e.g., temperature and electromagnetic radiation. This template-based defect approach is not limited to defects that may impact a MOS but can be used to define defect templates for several SPICE components, including defects at the block level. Listing 3.10 describes the defect templates for a MOS coded in Verilog-A. The complete list of defect templates proposed is available in Appendix A. The templates are defined inside a module named `TARGET_MOSFET_MODEL` that contains the fault-free case and ten defect models as envisioned by the IEEE P2427 draft standard. The Verilog-A module consists of a `generate-case` block driven by a `failMode` parameter, which activates the selected defect. The complete list of defects modeled within this Verilog-A module is: (0) fault-free, (1) drain open, (2) gate open, (3) source open, (4) bulk open, (5) gate-drain short, (6) gate-source short, (7) gate-body short, (8) body-drain short, (9) body-source short, and (10) drain-source short. Inside the `generate-case` blocks, the template can instantiate the fault-free model, which allows calling a native subcircuit model or a primitive from within the Verilog-A code. The defect template is free to play around with instance parameters of the fault-free instance, e.g., use the multiplier parameter `M` to mimic 1 out N capacitor open in an array. Normally, the defect template should reveal the same instance parameters as the original instance of the fault-free model, meaning that a given template can only be applied to models that provide a compatible set of instance parameters. In order to specify the fault-free model to use in the template, a pre-processor macro is used and is named `ORIGINAL_MOSFET_MODEL`. The construct with the `for()` loop is used to pass parameter values to the `genvar mode` in order to instantiate the different `case` blocks at compilation time. In addition, the defect template can instantiate other components to model the defect or by defining constraints on branch voltages and currents. For the injection, a `sweep` statement is used in the testbench to substitute the original instance of the fault-free model with an instance of the defect template. If the defect has a fault-free model associated with it, this substitution can be made at once at multiple places, and the defect activation can be done by setting the `failMode` parameters to the wanted value (one by one).

The main features of the proposed implementation can be summarized in four points. First, the defect templates can be made simulator-independent (depending on having the same instance parameters and model names for a component in multiple simulators). Second, the syntax is clean and simple, one template is one module, and each failure mode is a case statement. Third, by looking into the simulation outputs, it is simpler to assess which defect was injected by looking at a dedicated instance parameter value than by inspecting the master name. Moreover, with our approach, designers need to validate the behavior of the

```
1 <component anabasis_IEEE2427hard_m, defect_model ds_short__>
2   * Uses the following parameters:
3   * - MODE_<MM> with <MM> one of: 'AC', 'IC', 'DC', 'TRAN': selects how to set the internal probing switches.
4   *   MODE_<MM> = 1: activates the fault-free case during analysis 'MM'.
5   *   MODE_<MM> = 2: activates the faulty case during analysis 'MM'.
6   *   MODE_<MM> = 3: activates the S-parameter probing mode during analysis 'MM' (should be an AC-like analysis).
7   * - FAULT_VALUE: is the value of the faulty resistor in Ohms (may vary between 0 and infinity). Default is 100
        ohms
8   *   The value corresponding to the fault-free case is hard-coded as 1.0 TOhm.
9   * - FAULT_PORT<i>: defines the number of the S-parameter port instantiated at the fault location.
10  *defect_type = short
11  *insert_subckt_header
12  X_ORIGINAL port1 @port1 @port3 @port4 @subckt_name @subckt_parameters
13  X_FAULT @port1 @port3 RESFAULT FAULT_VALUE=FAULT_VALUE NOMINAL_VALUE=1.0E12 MODE_DC=MODE_DC MODE_TRAN=MODE_TRAN
        MODE_AC=MODE_AC MODE_IC=MODE_IC PORTZC=50 PORTNBR=FAULT_PORT1
14  *insert_subckt_footer
15 <endcomponent anabasis_IEEE2427hard_m
16
17 <component anabasis_IEEE2427hard_m, defect_model source_open_>
18  * Uses the following parameters:
19  * - MODE_<MM> with <MM> one of: 'AC', 'IC', 'DC', 'TRAN': selects how to set the internal probing switches.
20  *   MODE_<MM> = 1: activates the fault-free case during analysis 'MM'.
21  *   MODE_<MM> = 2: activates the faulty case during analysis 'MM'.
22  *   MODE_<MM> = 3: activates the S-parameter probing mode during analysis 'MM' (should be an AC-like analysis).
23  * - FAULT_VALUE: is the value of the faulty resistor in Ohms (may vary between 0 and infinity). Default is 1 Gohm
24  *   The value corresponding to the fault-free case is hard-coded as 1.0 mOhm.
25  * - FAULT_PORT<i>: defines the number of the S-parameter port instantiated at the fault location.
26  *defect_type = open
27  *insert_subckt_header
28  X_ORIGINAL @port1 @port2 DEFECTSIM_OPEN @port4 @subckt_name @subckt_parameters
29  X_FAULT @port3 DEFECTSIM_OPEN RESFAULT FAULT_VALUE=FAULT_VALUE NOMINAL_VALUE=1.0E-3 MODE_DC=MODE_DC MODE_TRAN=
        MODE_TRAN MODE_AC=MODE_AC MODE_IC=MODE_IC PORTZC=50 PORTNBR=FAULT_PORT1
30  *insert_subckt_footer
31 <endcomponent anabasis_IEEE2427hard_m>
```

Listing 3.11: Custom Defectsim templates for short and open fault.

templates only once time and not for each different simulator used. Last, AHDL-based defect templates can issue messages into the simulator log file, enabling custom optimizations.

Comparison with the state of the art defect templates

The state-of-the-art methodology to inject defects into transistor-level descriptions is to use the proprietary tool Tessent Defectsim provided by Siemens EDA. Defectsim templates depend on the syntax of the underlying simulator and have a complex syntax mixing TCL and SPICE. The defects are injected one at a time by creating a faulty subcircuit for each defect injected [89]. While the activations of these defects can be obtained only by using `alter` statements and not by using parametric sweep as with the proposed defect templates. Listing 3.11 describe with pseudocode the definition of two Defectsim templates used to inject two defects. The first code describes how to inject a short defect between two ports, and it uses many proprietary conventions. While the second code describes how to inject an open defect between two ports.

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

3.4.3 Manipulation framework

This section presents the language-independent framework that allows manipulating transistor-level descriptions easily. The entire framework is open-source and available on GitHub [6]. In Figure 3.18, a schematic structure of the manipulation framework is presented. The entire framework is written in C++, and all the functions are wrapped in Python to allow better usage. Several front-ends bring heterogeneous

[6] https://github.com/sydelity-net/EDACurry

descriptions into an in-memory description based on the AST generated by the parsing phase within the framework. These language descriptions that the framework reads as input can be written in Eldo, Spectre, or XML languages as output. A grammar for Eldo (the Eldo grammar also covers SPICE) and Spectre written in a format compatible with ANTLR4 are provided within the framework. After the parsing phase, the structure of the considered AST will be fully represented within the internal structure of the framework. This means that no information will be lost during this transcription, from the input description to the internal structure of the framework. The framework allows several manipulations of the internal structure, such as changing the master of a component or the subcircuit pins. After applying manipulation techniques, the internal structure of the framework is printed through several back-ends. It is possible to print the internal structure as Eldo or Spectre netlist or to XML or JSON files.

In the proposed workflow (see Figure 3.27 for the schematic flow), the framework is used to manipulate the netlist and the testbench (written in SPICE, Eldo, or SPECTRE) to add the lines of code required to inject the Verilog-A defect templates (proposed in Section 3.4.2). In detail, two manipulations are performed by the framework; first, by adding the defect parameters to the subcircuit under analysis as shown in Listing 3.13 for the OPAMP circuit. Second, by adding to the original testbench the Verilog-A defect templates by using the include command, the correct model instantiation, and the sweeping on the defect parameters.

3.4.4 Experimental validation

The defect templates proposed in Section 3.4.2 and the netlist manipulation framework were validated using the circuits in the analog testbench suite provided by the IEEE P2427 standard [35]. In particular, we show how it is possible to inject the defect templates automatically inside two MOSs inside the Operational Amplifier (OPAMP) circuit. The schematic of the OPAMP amplifier used for the experiments is shown in Figure 3.1, and it has six input pins and one output pin (both highlighted with a red label). The netlist description of the same circuit written with the SPECTRE language is presented in Listing 3.12. For each `pmos1` and `nmos1` inside the netlist, the relative parameters are simplified with a `<params>` keyword. The netlist is composed of fourteen MOS components, respectively seven MOS type-n and seven type-p.

The proposed MOS defect templates can be applied for each MOS that composes the netlist. To show the effectiveness of the proposed approach, the defect templates are applied on two MOSs, the `mpd11` and `mpd11` components. By using the framework infrastructure presented in Section 3.4.3, the original OPAMP netlist (see Listing 3.12) is instrumented automatically to include the parameters `failmode_mpd11=0` and `failmode_mpd12=0` and to change the model name for the MOS under test from `pmos1` to `anabasis_pmos1`. A sketch of the resulting netlist is shown in Listing 3.13, where these changes are visible for the components `mpd11` and `mpd12`.

Then, the next step before simulating the defect templates applied to different MOS components is to instrument the testbench. Starting from an existing testbench written in SPECTRE through the manipulation framework is easy to instrument the testbench with the additional lines required to perform the defect injection campaign. The final testbench used to perform the defect injection campaign on the MOS `mpd11` and `mpd12` components is shown in Listing 3.14. The transformations that the manipulation framework has applied in order to build this instrumented version are on line 6, where the defect templates written in Verilog-A are included with the `ahdl_include` command that allows to include HDL descriptions into SPECTRE or SPICE compatible code. Then, the `OPAMP1_WRAPPED` DUT is instantiated in line 23, with the associated parameters `failmode_mpd12` and `failmode_mpd12` necessary to inject the defect templates on the two MOS components. The last change produced by the manipulator framework is the most important to enable the defect injection campaign. In lines 31 and 35, the sweeping on the defect

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

Listing 3.12: OPAMP subcircuit modeled in SPECTRE.

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

Listing 3.13: OPAMP subcircuit modeled in SPECTRE that contains the transformations in order to be able to use defect injection wrappers.

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

Listing 3.14: OPAMP1 testbench written with the SPECTRE language.

parameters `failmode_mpd12` and `failmode_mpd12` are added going from 0 to 10 in order to simulate all the defect parameter templates for both the MOS components. The complete list of defects injected for each component guided by the defect template is: (0) fault-free, (1) drain open, (2) gate open, (3) source open, (4) bulk open, (5) gate-drain short, (6) gate-source short, (7) gate-body short, (8) body-drain short, (9) body-source short, and (10) drain-source short.

The transient simulation of the presented test case is performed by using the Siemens EDA Symphony simulator on a CentOS 7 machine. The resulting waveforms for the `out` pin of the OPAMP circuit are plotted

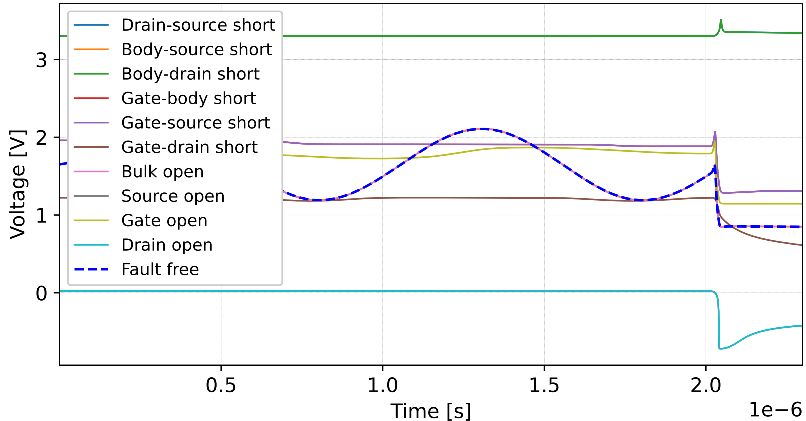


Fig. 3.30: OPAMP output waveforms for the different faults injected on the `mpd11` MOS component.

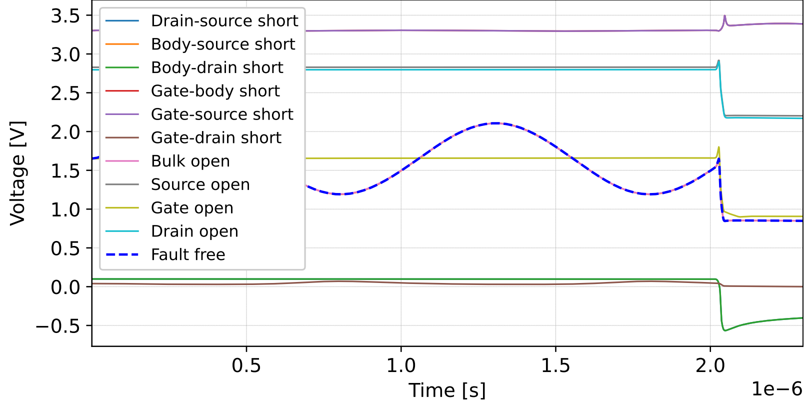


Fig. 3.31: OPAMP output waveforms for the different faults injected on the `mpd12` MOS component.

in order to compare the different defect templates injected. Figure 3.30 shows the fault-free waveforms and the faulty waveforms for each defect injected on the `mpd11` MOS component. While Figure 3.31 shows the resulting waveforms for the `mpd12` MOS component. The simplicity of the presented approach allows to compare the results easily and to apply further analysis on the faulty behaviors, like fault grouping techniques.

3.5 Analog fault grouping

Defect injection has become an indispensable tool for optimizing the diagnostic coverage of analog circuits to guarantee functional safety [126]. The promising approaches from the state-of-the-art aim at reducing the number of faults through techniques like *grouping* faults into classes and studying the *equivalence* between faults. As opposed to an analog circuit, in a digital circuit, checking the equivalence of faults is easier to perform due to the *discrete* nature of signals (i.e., they can be either 0 or 1). But with analog ICs, which are modeled using *continuous* values, equivalence between faults is more complex to be defined. The combination of the number of faults to consider and the length of analog simulations generates a fault-injection campaign that is very time-consuming and computationally expensive.

It has been found that several faults may induce similar faulty behavior at the circuit-block level. Identifying which group of faults induces the same failure mode allows the definition of fault-equivalence classes and reduces the simulation effort. With this approach, it is only necessary to check the coverage once for every failure mode at the system level. A complete characterization of faults, even at the circuit-block level, requires a significant computational effort [79]. It is common practice today to assess the equivalence of

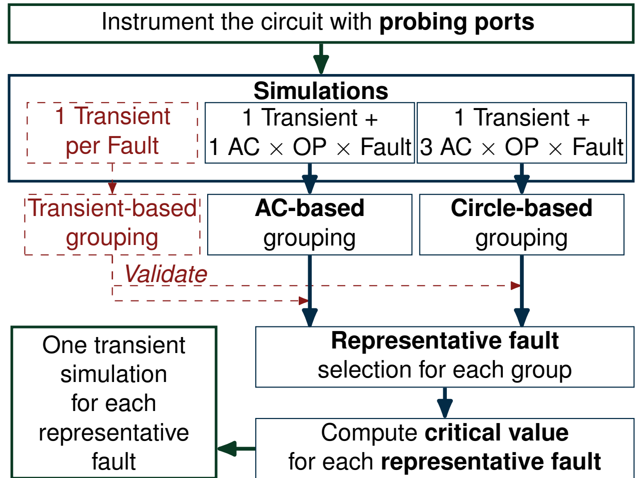


Fig. 3.32: Overview of the proposed methodology to group equivalent fault behaviors.

faults by comparing the results of test procedures applied in simulation to a CUT. This analysis typically requires running a combination of several DC and transient simulations, which are usually time-consuming activities. In this article, a predictive fault grouping based on faulty AC matrices is applied to group faults with equivalent behaviors. The proposed method is computationally cheap because it avoids performing DC or transient simulations with faults injected and limits itself only to faulty AC simulations.

Figure 3.32 depicts the proposed methodological flow. First, we instrument the circuit with probing ports required to extract the AC matrices. Then, we perform a single transient simulation interleaved at specific OPs by several AC simulations. We perform one AC simulation for the fault-free case and, at most, three simulations for each fault. The number of AC simulations for each fault varies from one to three based on the applied method (i.e., AC- or circle-based grouping). Three simulations are carried out to extract the data for the AC-based grouping: one fault-free transient simulation, and for each OP one faulty AC simulation with a critical fault parameter value (behave as faulty), and another AC simulation with a nominal parameter value (behave as fault-free). In fact, with the circuit shown in Figure 3.34, all faults could be instrumented at once, so the transient simulation is run once and covers all faults. Only the AC simulations would have to be repeated per each fault. The second proposed approach is circle-based grouping, which is based on the AC matrices and exploits the circle-fitting method to group the different faults. In this work, two grouping methods operating on AC data are compared: 1) grouping based on the comparison of feature vectors consisting of S-parameters and 2) grouping based on feature vectors constructed from the centers of the fault circles. The proposed fault grouping methods are validated with respect to a transient-based grouping. It is also possible to group the faults based on the result of the circle-fitting method, but computationally more expensive. The effectiveness of the proposed work is tested on circuits taken from the IEEE analog benchmarks [35].

3.5.1 State of the art and definitions

This section is subdivided into four parts: The *first* part briefly details the types of fault models affecting ICs described at the transistor level, as described in the IEEE P2427 draft standard [32]. The *second* part provides the necessary concepts regarding AC matrices. The *third* part explains how to instrument the circuit to retrieve the data required for the methodology. Finally, the *fourth* part introduces the state-of-the-art related to analog fault grouping.

Types of analog failure modes

There are usually two types of faults in analog descriptions: *soft* and *hard* faults [34]. A slight deviation of component parametric values to their nominal value, due to either systematic or random process variations, is called *soft* fault. While a change of the circuit's topology due to the presence of *primarily* short or open circuits between nets is called *hard* fault.

In this section, different faults are considered, in details these faults are grouped for each electrical component following the IEEE P2427 draft standard [32]:

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

AC matrices

The AC matrices are retrieved as the scattering parameters, also called S-parameter, during an AC simulation. These AC matrices describe the input-output relations between the ports in an electrical circuit. These relations are defined as a complex matrix (AC matrix) that shows the reflection/transmission characteristics (amplitude/phase) in the frequency domain [127]. Particularly, it becomes essential to describe a given network in terms of waves rather than voltage or current at high frequency. Thus, it is possible to analyze the circuit's frequency behavior through the S-parameters [128]. The numbering convention for S-parameters contains two numbers enclosed in round brackets and preceded by an `S`, e.g., `S(N1, N2)`. Here, `N1` identifies the port where the signal appears (the output), and `N2` identifies the port number where the signal is applied (the input). Starting from the AC matrices retrieved with the S-parameters, it is possible to extract the Y-parameters, Z-parameters [129], H-parameters, T-parameters or ABCD-parameters [130].

Circuit instrumentation

A transient simulation is performed for each fault, and depending on the behavior of our design, the circuit is linearized at one or more operating points. At each of these operating points, we pause the simulation and perform an AC simulation. During the AC simulation, standard S-parameter probes are used to retrieve the circuit's behavior at different frequencies. The S-parameter probes are connected to our design through the 3-terminal circuit detailed in Figure 3.33. When the internal switch is in the *first* positioned between `cut` and `env`, the S-parameter probe is not active, while when it is in the *second* position between `cut` and `gnd` the probe is active. When the *second* position is activated, the `cut` is isolated from the rest of the circuit, and through the stimuli generated by the probe, it is possible to extract the frequency behavior of the `cut`.

The 3-terminal circuit is useful for analyzing circuits' behavior. However, our objective is to analyze their faulty behavior. For this purpose, we developed the component shown in Figure 3.34, consisting of two switches connected through two pins inside the circuit. This component allows reproducing both faulty and fault-free behavior, as well as probing S-parameters. In detail, this component allows reproducing the faulty behavior by injecting a parametrized fault (middle), reproducing the fault-free behavior (upper), and

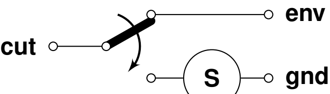


Fig. 3.33: A 3-terminal circuit for S-parameter extraction.

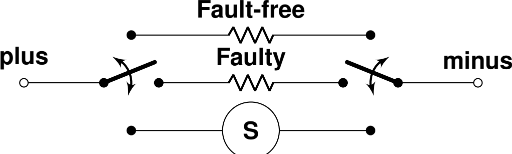


Fig. 3.34: Our implementation of an S-parameter port connected only with two terminals inside an analog circuit.

acquiring S-parameters through a probe (lower). Different configurations of the two switches are created to perform the different simulations on the CUT required for the grouping phases. When the two switches are *both*:

- in the *upper* position, the circuit does not alter the CUT behavior during transient simulation;
- in the *middle*, it injects a parametrized faulty resistor in parallel to an S-parameter probe, which is required during the AC simulation for extracting the faulty AC matrices;
- in the *lower* position, it extracts the S-parameters with the probe, saving the AC matrices at the point where the fault is injected.

The S-parameters are then used to calculate the critical value for a fault parameter.

State of the art of analog fault grouping techniques

Several analog fault simulation and grouping techniques are presented and compared with our methodology in this section.

In [72], the authors discussed fault grouping based on hierarchical clustering on the faulty transient waveforms. They showed that the grouping they realized allowed predicting the final classification of faults in functional safety categories reliably. This result constitutes the *golden reference* of the present article, as our method strives to reproduce the same grouping as obtained from transient waveforms but by alternative, much less computationally intensive means.

In [131], the authors show the limitations of the random sampling [132] for choosing the faults to be simulated in an analog circuit and highlight the necessity to consider the uncertainty of the fault parameter before simulating a fault. For that reason, in this work, we try to prioritize faults by applying an approximated grouping on the faults and taking the *central* component as representative fault for each group. With *central* component, we mean the one that minimizes the distances to all the other group members. We also analyze the uncertainty of the fault parameter value that the representative faults assume by calculating a *critical value* for it. The *critical value* for a fault leads to an AC parameter that has the potential to deviate the most from its fault-free value. In [66], authors presented a dynamic fault grouping algorithm for non-linear circuits. This methodology dynamically divides the list of faults into groups based on the transient behavior. Different time steps are used for different groups in simulation; however, the time step for faults within the same group stays the same. Another fault list compression technique is presented in [73], which uses stratified fault grouping for the identification of representative faults.

Most of the techniques mentioned above are based on transient simulations, which are well-known for being time-consuming. Other techniques have explored different solutions that tend to reduce the number of transient simulations. Frequency-based analysis was presented in papers like [78] and [133]. In [79], authors presented a fault clustering technique combining faulty DC OP and frequency domain analysis. That technique requires N transient simulations (i.e., one for each fault), and then, at each OP, they perform an AC analysis. With this approach, we extract faulty AC matrices at the fault-free OP, not at the faulty one. The difference is that for N faults, we need to do one transient simulation to generate the OPs, then, at each OP, we perform the AC analysis with the faulty matrix.

3.5.2 Impact of a fault on the AC matrices

A fault in an analog circuit can lead the circuit to assume behaviors different from the nominal one or behave like the circuit without the presence of the fault. To identify these abnormal behaviors of the CUT, we strive to characterize faults in a generic manner independent of a given test method or performance metric. We want to characterize *raw behavior* such that the grouping result would remain valid whatever the situation the circuit is put in later. It is known that the collection of AC matrices at various operating points is sufficient to characterize the non-linear behavior of an analog circuit (indeed, this is the fundamental principle of SPICE solvers). Like in [79], we use AC matrices at several operating points to characterize the circuit behavior in a generic manner.

Let us take a circuit with N ports represented by its scattering matrix: $b_N = S_N \cdot a_N$, where a_N and b_N are vectors of incoming and out-going (reflected) traveling waveform response and S_N the scattering matrix. We can formulate a scattering matrix S_{N+1} for the circuit with an additional port such that the fault is externalized as detailed in the Equation (3.1):

$$\begin{bmatrix} b_1 \\ \vdots \\ b_N \\ b_X \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} S_{PP} \end{bmatrix} & \begin{bmatrix} S_{PF} \end{bmatrix} \\ \begin{bmatrix} S_{FP} \end{bmatrix} & \begin{bmatrix} S_{FF} \end{bmatrix} \end{bmatrix} \cdot \begin{bmatrix} a_1 \\ \vdots \\ a_N \\ a_X \end{bmatrix} \tag{3.1}$$

where a_X is defined as $a_X = \varGamma \cdot b_X$. The fault model is represented by the Equation (3.2):

$$\varGamma_f = \frac{1 - g_f \cdot Z_c}{1 + g_f \cdot Z_c} \tag{3.2}$$

where g_f is the fault parameter, and Z_C is the reference impedance. By substituting a_X and solving for b_X we obtain an equation for the external S-parameters of the circuit in the function of the fault parameter $\varGamma_f \in]-1, +1[$ as described in the Equation (3.3):

$$S_N(\varGamma_f) = S_{PP} + S_{PF} \cdot (1 - \varGamma_f \cdot S_{FF})^{-1} \cdot \varGamma_f \cdot S_{FP} \tag{3.3}$$

Equation (3.3) is also valid in case the fault network is a multi-port. Focusing on a single element of the matrix S_N we obtain the Equation (3.4):

$$S_{i,j}(\varGamma_f) = S_{PPi,j} + \frac{\varGamma_f \cdot S_{Xi,j}}{1 - \varGamma_f \cdot S_{FF}} \tag{3.4}$$

where $S_{PPi,j}$, $S_{Xi,j}$ and S_{FF} are complex constants, independent of the value of $\varGamma_f$. For physical realizations, $\varGamma_f$ never reaches $+1$ or -1, which corresponds to the ideal open or the ideal short fault. Equation (3.4) shows that some values of $\varGamma_f$ will result in a maximal deviation of the $S_{i,j}$, representing the so-called *critical value*.

Impact of a fault on the admittance matrices

The admittance matrices can be easily calculated from the AC matrices (composed of complex values) retrieved through the S-parameter ports [134]. Assume a circuit with N pins represented by its admittance matrix: $I_N = Y_N \cdot V_N$ where I_N and V_N are vectors of pin currents and pin voltages response. Imagine one branch inside with a faulty conductance g_f between nodes p and n. It is possible to formulate a nodal

admittance matrix $Y_N + 2$ the circuit with two additional pins such that the fault is externalized as described in the Equation (3.5).

$$\begin{bmatrix} I_1 \\ \vdots \\ I_N \\ I_p \\ I_q \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} Y_{PP} \end{bmatrix} & \begin{bmatrix} Y_{PF} \end{bmatrix} \\ & \\ \begin{bmatrix} Y_{FP} \end{bmatrix} & \begin{bmatrix} Y_{FF} \end{bmatrix} \end{bmatrix} \cdot \begin{bmatrix} V_1 \\ \vdots \\ V_N \\ V_p \\ V_q \end{bmatrix} \tag{3.5}$$

The previous system can be simplified taking the Equation (3.6).

$$V_f \triangleq V_p - V_q \text{ and } I_f \triangleq 1/2(I_q - I_p) \tag{3.6}$$

Then substituting the Equation (3.6) in the Equation (3.5) are derived the Equation (3.7) that represent the circuit behavior.

$$\begin{bmatrix} I_1 \\ \vdots \\ I_N \\ -If \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} Y_{PP} \end{bmatrix} & \begin{bmatrix} Y'_{PF} \end{bmatrix} \\ & \\ \begin{bmatrix} Y'_{FP} \end{bmatrix} & \begin{bmatrix} Y'_{FF} \end{bmatrix} \end{bmatrix} \cdot \begin{bmatrix} V_1 \\ \vdots \\ V_N \\ V_f \end{bmatrix} \tag{3.7}$$

The fault model is defined in the Equation (3.8).

$$I_f = g_f \cdot V_f \tag{3.8}$$

The impact of the fault on the matrix Y_N can be obtained by solving for V_f as depicted in the Equation (3.9).

$$Y_N(gf) = Y_{PP} + Y'_{PF} \cdot (gf - Y'_{FF})^{-1} \cdot Y'_{FP} \tag{3.9}$$

The previous equation means that: at an frequency, each element of the admittance matrix can be formulated with the Equation (3.10).

$$Y_{i,j}(gf) = Y_{PPi,j} + \frac{Y_{Xi,j}}{g_f - Y'_{FF}} \tag{3.10}$$

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

3.5.3 Predictive fault grouping

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

AC-based grouping

The paper of [79] introduced the idea of fault grouping based on the AC matrices response. It combines the analysis of faulty OP in DC simulations and faulty AC simulations to group the faults and compare them with the transient results. In this approach, to perform the clustering `N` transient simulations (each time the fault is injected) are required, and then at each OP an AC simulation is performed. Conversely, our approach is based on the extraction of the AC matrices at fault-free OP. Hierarchical clustering is used because it reveals the distance structure between the faults. Let us consider `N` faults; we need to do one

transient simulation to generate the OPs, then at each OP, do the AC simulation with the fault and the probe injected. The AC-based grouping relies on feature vectors constructed from the elements of the complex-valued scattering matrices, extracted at various frequencies and time points.

After collecting all the S-parameter the data are processed using a clustering technique. The metric used to subdivide the cluster is depicted in the Equation (3.11) and represents the Euclidean distance between two S-parameter matrices:

$$d(\mathbf{S_A}, \mathbf{S_B}) = \sqrt{\sum_{i,j,k,l} \left(S_{A(i,j)}(t_k, f_l) - S_{B(i,j)}(t_k, f_l)\right)^2} \tag{3.11}$$

where S_A and S_B are two different S-parameter matrices relative to two distinct faults.

Circle-based grouping

In the world of IC testing, it is known that the AC matrix elements at a given frequency and operating point describe circular trajectories in the complex plane when the fault parameter is varied over a wide range of values [135]. These circles represent the locus of AC matrix element values for the distribution of fault parameter values and can be used to represent a fault characterized by an uncertain fault parameter uniquely. Hence the idea to use the distance between fault circles to compare uncertain faults.

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

The circle-fitting method proposed in this article allows proving that two faults are equivalent and enables fault clustering by looking at the feature vectors constructed from the centers of the fault circles. Our implementation of the circle-fitting method is based on the Hyper Circle algorithm [136]. At least two faulty AC simulations must be performed with different fault parameter values plus the AC fault-free simulation to retrieve the necessary data to apply the Circle-based grouping. These three AC simulations allow to identify at least three points in the *Cx* plane on which the circle-fitting algorithm is applied. After identifying all the circles in the AC faulty matrices, the clustering is performed based on the metric of Equation (3.12):

$$d(\mathbf{C_{SA}}, \mathbf{C_{SB}}) = \sum_{i,j,k,l} \Big| cc(S_{A(i,j)}(t_k, f_l, \Gamma_A)) - cc(S_{B(i,j)}(t_k, f_l, \Gamma_B)) \Big| \tag{3.12}$$

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

Transient-based grouping

This section shows how to perform a clustering of waveforms (`VOUT` waveform) based on their trajectory. It is well known that it is possible to group faults by running transient fault simulations and then group the waveforms [72]. However, this technique is time-consuming and requires a sensible amount of computational resources, so it is presented in this section for the sole purpose of validating the proposed methodology (see Figure 1.1). Basically, we are comparing waveforms for various faults using a modified

Euclidean distance measure over the space of functions applied to the simulation time interval. This measure is used as a matrice to perform the clustering.

Computing the distance measure between two waveforms W_A and W_B is performed with the Equation 3.13:

$$d(W_A, W_B) = \frac{t_{e,ff} - t_0}{t_{e,fy} - t_0} \cdot \int_{t_0}^{t_{e,fy}} (W_A(t) - W_B(t))^2 \cdot dt \tag{3.13}$$

with t_0 the start-time of the simulations, $t_{e,ff}$ and $t_{e,fy}$ the end times of the faulty and fault-free simulations with the same stimuli (sometimes the simulator aborts the faulty simulation prematurely – this is compensated by the fraction in front of the integral). The innovation we proposed with Equation 3.13 is the calculus of the distance between two waveforms, by considering a *modified Euclidean distance* that is integrated over a weighted time correction factor. This correction factor used in the formula allows for penalizing simulations that did not run till the end.

3.5.4 Methodology validation

In this section, the proposed methodology flow is applied to the OpAmp circuit shown in Figure 3.1, retrieved from the analog-benchmark circuits collection [35]. The different grouping techniques are compared to show the potentialities of the AC-based and the Circle-based ones.

Circle-fitting method

The Circle-based grouping is relying on the circle-fitting applied to the faulty AC matrices. In Figure 3.35 some circles are plotted for the fault `D20`, that is, a MOS drain-source short on the transistor `MNM12` located in the OpAmp circuit, in the *Cx* plane. The circles are plotted for three different frequency values: $1Hz$, $10kHz$, and $10MHz$. In these plots, blue lines are the fault trajectories that are a function of the fault parameter value. One blue curve is related to one time point.

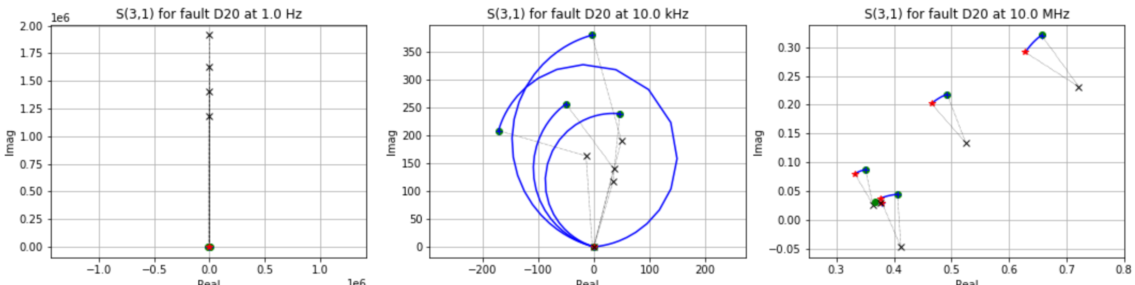


Fig. 3.35: AC trajectories of the fault D20 (MOS drain-source short on the transistor `MNM12`) at three frequencies and four operating points.

The red star identifies the faulty case. The green ring is the fault-free case retrieved from the fault-free simulation with no fault probe injected. From these plots is simple to identify well-defined circles in the middle plot.

Figure 3.36 shows the distribution of the fault parameter in the complex plane for four different faults, two different operating points (columns), and different frequency values, 10.0 MHz and 10.0 GHz (rows) for the OpAmp circuit. These four faults plotted are open and short faults injected in four MOS injected one fault at a time. Only three complex values are required to identify a circle for a single fault with a fixed frequency [135]. Starting from these complex values, the circle-fitting algorithm based on the Hyper Circle

Algorithm allows finding the center of the circle and the radius. These plots are generated through the data computed with the circle-fitting algorithm [136].

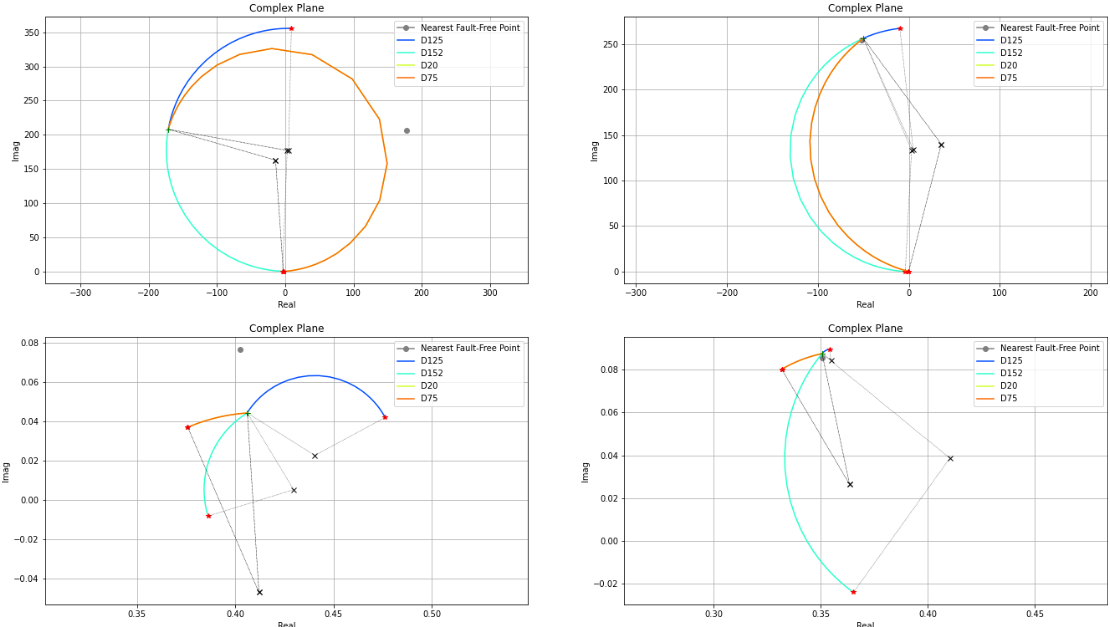


Fig. 3.36: Circles generated for the faults #D125, #D152, #D20 and #D75 relatives to the OpAmp circuit. These circles are generated with the circle-fitting algorithm for two operating points (columns) and two frequency points (rows).

Comparison between the different grouping techniques

Figure 3.37 and Figure 3.38 present the group-membership correlation matrices of the faults. An element at position `(i, j)` is set to 1 if the faults corresponding to indices `i` and `j` belong to the same group and to 0 otherwise. We generated middle plots of both Figure 3.37 and Figure 3.38 called "Differences", by an element-wise application of the xor logic operator to the two outer matrices. The xor operation will produce a 1 (non-zero value) if only if the two input values are different. As the differences increase, the middle plot will have more black dots, while if the differences are fewer, the plot will have fewer black dots.

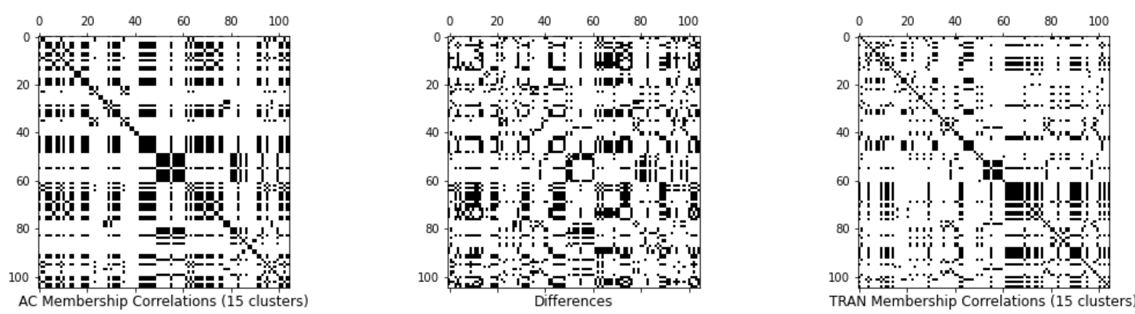
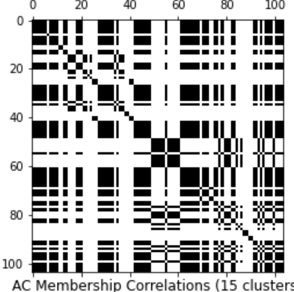
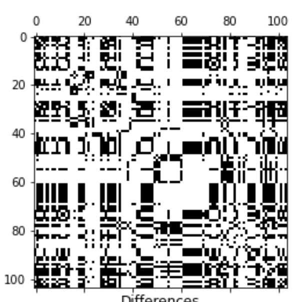


Fig. 3.37: Comparison between AC and transient-based grouping (with 15 clusters).





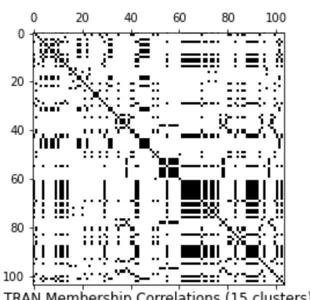


Fig. 3.38: Comparison between the circle and transient-based grouping (with 15 clusters).

AC-based vs transient-based grouping

In Figure 3.37 on the left is plotted the AC membership correlation based on the S-parameters (with 15 clusters), while, on the right, the transient membership correlation is plotted (15 clusters). As shown by the plot in the middle, the error in the group membership correlations is around 23%, meaning that the overall group structure has many similarities. This correlation error allows us to state that the proposed technique is robust and efficient, as it allows us to perform only AC simulations, saving time and computational resources.

Circle-based vs transient-based grouping

In Figure 3.38 on the left is plotted the AC membership correlation based on the Circle-fitting method (with 15 clusters), while, on the right, the transient membership correlation is plotted (15 clusters). As shown by the plot in the middle, the error in the group membership correlations is around 34%, meaning that the overall group structure has fewer similarities with respect to the grouping based on the S-parameters. This higher error is mostly due to numerical errors that arise in the computation of circles, especially for low frequencies, because the imaginary part of the AC matrices tends to be zero.

Critical parameter value of a fault

Table 3.1: Critical values at different frequency points for the most significative OPAMP circuit faults and their group.

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

After clustering on the AC matrices (faulty and non-faulty) and selecting a representative fault for each cluster, chosen in the middle of the cluster, it is necessary to simulate these faults in order to understand the behavior of our design in the presence of faults. To simulate these faults and produce alterations that deviate significantly from the fault-free behavior, it is necessary to inject each fault with an appropriate value. This value is the resistance value that is associated with the resistance used to model the fault. The

critical values and the fault grouping are reported in Table 3.1 for a subset of the OpAmp' faults. Table 3.1 reports the fault group, the nominal value that allows the fault to behave as fault-free, and the critical values calculated for three different frequencies: $10kHz$, $10MHz$, and $10GHz$. These critical values are computed by studying the admittance at the fault location with the fault disabled (switch placed in the third position, see Figure 3.34). This critical value means that the sensitivity of the AC parameter to the fault parameter is maximal. This means that for any other values, the sensitivity will be lower. It is crucial to identify this critical value to avoid injecting an undetectable fault that behaves like a faulty-free. The critical value for a given frequency is extracted from the admittance matrix, by looking for those values that have the modulus of the imaginary part closest to zero. The critical value (i.e., g_f) will be the real part of those values. If the fault-free value of a fault parameter is far from the critical value, then the circuit has a high tolerance against faulty values. Instead, if the faulty-free value is close to the critical value, then the circuit is susceptible to any fault.

4

Multi-domain fault modeling

The industrial field undergoes continuous and rapid evolution under the impulse of *Industry 4.0* innovations, granted by increasing precision of the design and simulation of the models. As the complexity of these models grows, increasing accuracy is required to be effective for industrial production [137] In the engineering world, it is also necessary to deal with problems caused by faults that could happen unpredictably due to different causes. Due to these unexpected failures, the ability to simulate entire production systems, or single machinery, before building them is becoming strategic. In the industrial environment, systems composed of many different parts belonging to different physical domains are involved. Considering the number of failures that could occur, it is essential to prevent dangerous situations by taking fault-prevention actions. As part of the functional safety of industrial machinery, simulating these systems is essential to ensure their proper operation. Once the simulated model is built, fault injection is performed to understand how the behavior of the system, or individual component, changes with respect to a given fault. This procedure, called *fault injection*, highlights new vulnerabilities or flaws that could compromise the safety of a system. However, in the context of Cyber-Physical System (CPS) (i.e., multi-domain systems), how to perform *fault injection* differs depending on the part of the system under analysis [138]. Considering an electromechanical system, the fault models injected into the electrical part will differ from purely mechanical ones. Moreover, techniques change also between different physical domains (e.g., electrical, mechanical, thermal). Overall, *fault injection* in analog and digital electrical models is the state of the practice to ensure functional safety, as mentioned by the ISO 26262 standard for automotive functional safety [9, 8]. After the publication of the ISO 26262 standard, it became apparent that in the analog field, there was a lack of precise standardization of all the faults necessary to apply fault injection procedures. For this reason, an IEEE workgroup is working on building the standard P2427 that aims to standardize all possible analog faults defined at transistor level [32], currently a draft standard.

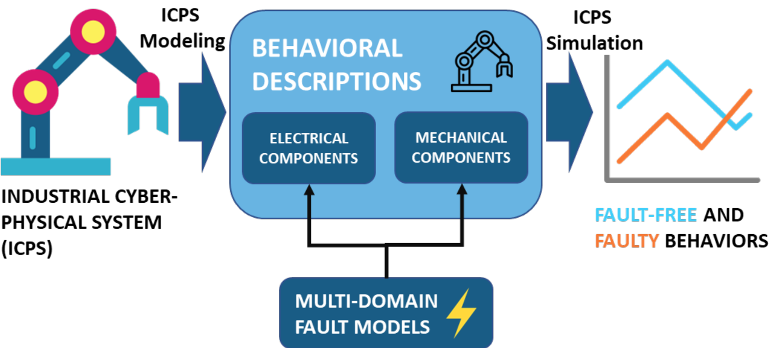


Fig. 4.1: Overview of the proposed multi-domain fault injection into a CPS.

Therefore, unlike the electrical domain, there is no large use of fault injection techniques in the mechanical one at the state of the practice. All the methodologies proposed in this chapter are based on the concepts explained in Chapter 3 related to fault modeling and injections in analog circuits. Extending the fault injection to the mechanical field could improve the evaluation of safety mechanisms in different faulty conditions. Primarily because of the difficulty in categorizing mechanical components and the working conditions of the machinery. Fortunately, some *analogies* between systems belonging to different physical domains utilize the same differential equations. The following sections show that there are *analogies* between mechanical and electrical systems [139]. The second part of this chapter aims to harness these analogies to model faulty behaviors of a generic system, regardless of the underlying physical domain Section 4.3. In this way, it would be possible to use the advanced fault injection techniques of the electrical domain in others, e.g., the mechanical one. Figure 4.1 shows an example of a faulty CPS, where different types of faults can lead to altering the arm manipulator functionality in many ways. Working at high temperatures, at a non-stop pace, in an unstable environment subject to movement and vibration (because it is not properly fixed to the ground) is very different from working under optimal conditions. State-of-the-art mechanical failure studies are focused on detection based on frequency analysis of vibration [140, 141, 142], the oil level in the system, noise emissions, and thermal dissipation [5], but there are no specific works related to the fault injection process applied to multi-domain systems.

All the proposed methodologies proposed in this chapter are tested on simple models but are developed to be scalable to more complex systems. For example, the automatic tool that is able to inject multi-domain faults into a Verilog-AMS description containing a behavioral system representation is developed to work with designs that combine more than one component and for supporting different classes of faults, comprises sporadic or periodic events that can be controlled from the testbench.

This chapter is organized as follows:

- Section 4.1 presents general concepts that enable to understand the proposed methodologies;
- Section 4.2 describe a general flow that allows representing a mechanical system with its electrical equivalent;
- Section 4.3 proposes a new mechanical fault taxonomy inferred by exploiting the physical analogies;
- Section 4.4 presents different mechanical models analyzed by injecting the faults proposed in the new mechanical fault taxonomy;
- Section 4.5 describe the tool built to enable automatic fault injection into multi-domain models written in Verilog-AMS and presents the problem of the testbench qualification in the context of mechanical fault injection;
- Section 4.6 proposes a new thermal fault taxonomy inferred by exploiting the physical analogies.

4.1 State of the art and definitions

This section describes the background related to the main concepts used to build the multi-domain fault injection techniques, e.g., multi-domain physical modeling, the Verilog-AMS language, mechanical faults, electrical faults, behavioral fault modeling, fault injection, Failure Mode and Effect Analysis (FMEA), and the HIFSuite framework.

4.1.1 Multi-domain physical modeling

Two approaches exist at the state-of-the-art to represent and simulate physical systems. One alternative is to adopt graphical tools, e.g., Modelica-based tools and Simulink/Simscape. With this solution, the system

is constructed through the connection of predefined blocks belonging to domain-specific libraries. The internal dynamics of such predefined blocks are not visible, and the underlying equations are visible only to the solver that computes system evolution over time. Blocks are nonetheless parametrized and thus can be customized by the designer.

The complementary approach is to adopt Hardware Description Language (HDL), like Verilog-AMS, VHDL-AMS, and SystemC-AMS. Such solutions require explicitly modeling the system evolution as differential and algebraic equations enriched with the application of energy conservation laws. This imposes a higher effort during model construction. On the other hand, the designer can choose the level of detail of the model, and it becomes possible to inject faults and faulty behaviors. A HDL is a programming language developed to ease the formal description of digital circuits as well. Compared to usual programming languages, these languages allow describing the connection of the different parts, distinguishing structural, behavioral, and data flow components, and to consider timing and delay propagation concepts. Instead of generating a computer executable file, the HDL compilers provide a netlist ready to be translated into an integrated circuit; the netlist creation process is called synthesis. A hardware description can be analyzed and tested using specific software. In fact, the main feature of the hardware descriptions generated with these HDLs is that they can be synthesized and simulated virtually for testing purposes. This procedure is useful both for speeding up the development and final release and for monetary savings in case of design errors. The most popular HDLs languages are VHDL and Verilog.

Verilog-AMS Language: disciplines and natures

The Verilog-AMS language was created as an extension of the Verilog language for the *digital* part and Verilog-A for the *analog* part. Verilog-AMS, in fact, combines these two languages and allows to represent models that have a digital part that triggers the analog part and vice-versa [117]. This language can be used in the same way as VHDL-AMS; in fact, they share the same functionality [143]. Through the Verilog-AMS language, it is possible to model analog models described at the behavioral level that communicates with digital designs through the language's own functions, e.g., `timer()`, `cross()` functions.

Table 4.1: Verilog-AMS disciplines

Name	Potential/Across	Flow/Through
Logic	-	-
Electrical	Voltage	Current
Voltage	Voltage	-
Current	-	Current
Magnetic	Magneto Motive Force	Flux
Thermal	Temperature	Power
Kinematic	Position	Force
Kinematic_v	Velocity	Force
Rotational	Angle	Angular Force
Rotational_omega	Angular Velocity	Angular Force

These functions allow, for example, to activate a timer inside an analog design or to activate a cross() routine when the monitored function crosses the zero value of magnitude. Among the many innovative

Table 4.2: Verilog-AMS natures

Name	Units	Access	Abstol
Voltage	V	`V`	10^{-6}
Current	A	`I`	10^{-12}
Charge	C	`Q`	10^{-14}
Flux	Wb	`Phi`	10^{-9}
Magneto_Motive_Force	A-turns	`MMF`	10^{-12}
Temperature	$^{\circ}K$	`Temp`	10^{-4}
Power	W	`Pwr`	10^{-9}
Position	m	`Pos`	10^{-6}
Velocity	m/s	`Vel`	10^{-6}
Acceleration	m/s^2	`Acc`	10^{-6}
Impulse	m/s^3	`Imp`	10^{-6}
Force	N	`F`	10^{-6}
Angle	rad	`Theta`	10^{-6}
Angular_Velocity	rad/s	`Omega`	10^{-6}
Angular_Acceleration	rad/s^2	`Alpha`	10^{-6}
Angular_Force	Nm	`Tau`	10^{-6}

features of this language exist the support for models described across different physical domains, e.g., electrical, mechanical, magnetic, *etc*. Conservative domains are supported by defining the *potential* and *flow* variables associated with each physical domain.

For each domain, in fact, it is defined a *discipline* (see Table 4.1), which represents a physical domain and several *nature* (see Table 4.2) associated with each physical domain [144], e.g., for the electrical domain we have defined as natures the voltage and the current, accessible through the functions `V()` and `I()`. A nature is a collection of attributes that are shared by a class of signals. The table attributes include the units (units), the name used when accessing the signal (access), and absolute tolerance (abstol). By convention, the name of a nature begins with a capital letter. The table lists the natures included in the file `disciplines.vams` of the Verilog-AMS language. With SPICE-based languages, it is possible to simulate Verilog-AMS models into a transistor-level design, this implies that it is possible to control the simulation through ad-hoc testbenches written with specific control functions.

4.1.2 Mechanical Faults

Nowadays, there are still no state-of-the-art methodologies for the prevention of mechanical breakdowns of industrial components. This may also be due to the lack of a universal mechanical fault classification, i.e., one that is applicable to any mechanical component. In fact, due to precise classification, it is possible to test a system by injecting faults, studying its behavior, and comparing it with the fault-free system. However, the taxonomies proposed so far concern specific models: the faults are closely related to the structure of the machinery, so fault injection cannot be performed automatically.

Fault injection in analog and digital electrical models is the state of the practice to ensure functional safety, as mentioned within the ISO 26262 standard for automotive functional safety [9]. Through fault injection, it is possible to verify whether the system has been properly designed to ensure a correct response

in case of failure or defect. After the publication of the ISO 26262 standard, it became apparent that in the analog field, there was a lack of precise standardization of all the faults necessary to apply fault injection procedures. For this reason, an IEEE workgroup is building the standard P2427 that aims to standardize all possible analog faults defined at transistor-level [32]. In contrast, in the mechanical domain at the state of practice, there is no concrete use of fault injection techniques. This is related to the difficulty in categorizing components (i.e. translational and rotational) and the working conditions of the machinery. Working at high temperatures, at a non-stop pace, in an unstable environment subject to movement and vibration (because it is not properly fixed to the ground) is very different from working under optimal conditions. In fact, state-of-the-art mechanical failure studies focus on detection based on frequency analysis of vibration [140, 141, 142], the oil level in the system, noise emissions, and thermal dissipation [5], but not mention the mechanical fault injection. The most well-known causes of failure are changed operating conditions (e.g., higher load, misalignment, unbalance), altered intrinsic component parameters (e.g., increased friction due to dirt or high temperature), and geometric properties that are different from normal (e.g., a broken gear tooth). Failures investigated through analyses based on these causes are wear, stress-strain, breakage, damaged motor, and incorrect trajectory [145]. However, these failure models highlight a failure window that is too limited to be effective. In addition, there is no one-to-one correlation between the component and failure pattern. Table 4.3 presents some classical mechanical failure modes [146].

Table 4.3: Mechanical failure mode taxonomy.

Mechanical failure	Definition
Buckling	Deformation due to a load or force exerted parallel to the axis of the body
Corrosion	Geometric alteration due to a chemical reaction
Creep	Plastic deformation
Ductile deformation	Deformation for ductile materials
Fatigue	Fatigue due to continuous strain. This may result in cracks or breaks if deformation cycles increase
Fretting	Corrosion due to the adhesion of two solid surfaces (ex: two different metals)
Galling	Gripping, i.e., surfaces that are ruined by rubbing against each other due to different speeds
Seizure	Most severe gripping stage, the two surfaces are no longer separate
Impact	Deformation or fracture or fretting due to impact
Radiation	Exposure to nuclear radiation that changes the properties of the system
Rupture	Separation of parts of the same component
Spalling	Splintering of a surface that deteriorates the function of the component
Wear	Various kinds of wear: abrasive, adhesive, corrosive, plastic deformation, impact, surface fatigue

Extending the fault injection also to the mechanical field can improve the evaluation of safety mechanisms and highlight new vulnerabilities. In literature, few works exist that analyze multi-discipline faults, as it is challenging to model faults suitable for any physical model representation. It is essential to note that faults that can be modeled in a simulated environment are unlikely to cover all possible faults in the real world: specific physical phenomena affect each domain in a different way, and this requires taking into

account real-world facts. There are studies describing the injection of mechanical failures using specific tools [145, 147, 148]. The main limitation of all cited approaches is the difficulty of reproducing the alterations by using tools that rely on libraries of predefined components, whose internal dynamics are not accessible from users, and thus it is challenging to alter with faults.

4.1.3 Electrical Faults

Faults represent wrong behaviors of a system that can happen for multiple reasons, like device aging, the breaking of an internal component, a digital failure, or a production defect. Functional safety studies how it is possible to improve the security of a device by using adequate security systems. A fault is a *saboteur* if it consists of a new component that alters circuit behavior by injecting a new set of equations describing the faulty behavior. The fault is a *mutant* when it is a mutation of an existing component, e.g., a change in the nominal value of a parameter or a modification that models an abnormal behavior in the presence of faults. In contrast to the mechanical domain, in the electrical world, several failure models are part of the state of practice. The most common analog electrical faults, i.e. faults that are injected within transistor-level descriptions, are explained below.

The principal classes of electrical faults are: short, open, parametric, and sources. In detail:

- *Short* fault in an analog circuit is equivalent to bridge fault (see Figure 4.2-a). In a bridge fault, we have a connection not intended by the manufacturer between any two points of our circuit, while a short fault in a similar way is injected on a single component of the circuit; in this way it is sure that the two points of the circuit are close to each other. Generally, short faults are injected through a small resistance in parallel to a component.
- *Open fault* are related to a lack of a connection between two points in the circuit; as a result, the electrons flowing between the two points in the circuit are no longer free to pass from one side to the other. Therefore, an open fault is represented as a large resistor, which prevents the passage of current and is injected in series before or after a component. (see Figure 4.2-b).
- *Current source* faults are related to extra-currents injected in a circuit, usually, these controlled sources are modeled as pulses and injected in parallel with a component (see Figure 4.2-c).
- *Voltage source* faults are related to extra-voltage injected in a circuit, usually modeled as pulses and injected in series with a component (see Figure 4.2-d).
- *Parametric* faults are related to the variation of a parameter inside to a model. It does not refer to an external factor affecting the behavior of the circuit, but an alteration of a parameter of the original system itself. For example, a parametric fault might be a smaller resistance value in a circuit due to errors in manufacturing.

4.1.4 Behavioral fault modeling

Modeling and simulation of a Cyber-Physical System are key steps for ensuring the functional safety of such systems. At the state-of-the-art, there are two ways to represent and simulate physical systems: graphical tools, e.g., Modelica-based tools and Simulink/Simscape, and adopting description languages, like Verilog-AMS, VHDL-AMS, and SystemC-AMS. With the first solution, the system is built by connecting pre-defined blocks belonging to domain-specific libraries. The internal dynamic of such pre-defined blocks is not visible, and the underlying equations are visible only to the solver that computes system evolution over time. Blocks are nonetheless parametrized and thus can be customized by the designer. The second solution requires modeling the system evolution explicitly as differential and algebraic equations enriched with the

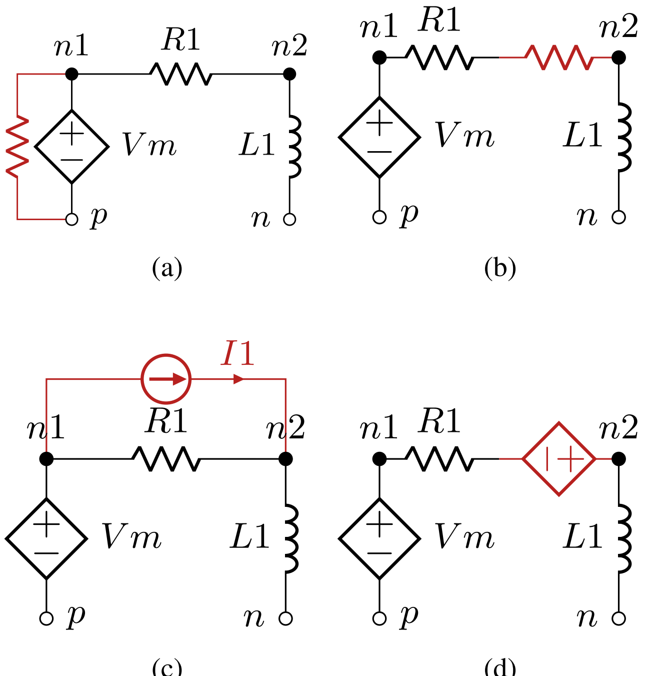


Fig. 4.2: Fault locations in a simple electrical circuit.

application of energy conservation laws. This imposes a higher effort during model construction. On the other hand, the designer can choose the level of detail of the model, and it becomes possible to inject faults and faulty behaviors at any level of detail.

A fault represents a wrong response in the system behavior because of multiple scenarios, i.e. material aging, strain or breaking of an internal component, a production defect, or a digital failure [149, 150]. Functional safety studies how it is possible to guarantee the functionalities of machinery in the presence of failures by exploiting different techniques. One of these techniques is *fault injection*, formally described in the ISO 26262 [9] standard for the electrical domain [32]. For the electrical domain, there are well-defined fault models and injection techniques that specify where and how to inject them into a transistor-level circuit [8].

Instead, the fault models and injection techniques are not advanced as the electrical ones in the mechanical domain. Fault taxonomies that consider the physical aspect of mechanical systems are listed in [151, 146]. However, it is possible to model these physical faults only if the model considers the geometry of the system and the behavior around it. By abstracting from the physical level to the behavioral level in the mechanical domain, the number of works that analyze multi-domain faults remains limited. Represent faults suitable for any physical model, and any level of abstraction is complex. Moreover, a fault modeled in a simulated environment is unlikely to cover all possible physical variations in the real world. The accuracy of these fault injection techniques concerning real-world physics depends on the model's level of detail.

4.1.5 Fault Injection

The fault injection is the basis of the safety analysis of complex systems, as described in [8] and in the *ISO 26262* standard [9]. ISO 26262 focuses mainly on the electrical domain, stating that functional safety must be ensured for all electronic components embedded into an automotive system. In this regard, fault

injection allows the verification of the effectiveness of all the safety mechanisms and proves the correct implementation of all the safety requirements.

The classification described in the previous section implies that a fault is injected on a single branch of the electrical circuit. A branch is identified by a pair of nodes: for example, in Figure 4.2, the branch of *R1* is identified by the tuple (`n1, n2`). Injecting a fault can be quite straightforward if you follow some simple steps: the largest difference is whether the faults are injected in series to the branch (open and voltage source) or in parallel (short and current source). Given a branch (`n1, n2`):

- if the fault is injected in series, a new temporary `nt` node must be created. Then, the fault (a strong resistor in case of the open; a voltage generator otherwise) is injected in the first sub-branch (`n1, nt`), while the branch (`nt, n2`) is the same as the original one.
- if the fault is injected in parallel, the fault (a weak resistor in case of short, a current generator otherwise) is injected directly on a new branch identified by the same nodes (`n1, n2`).
- while regarding, parametric failures, it is sufficient to change the coefficient value of the component that has to be failed.

Given these simple procedures, the process can be automatized to inject each type of fault on each branch. The injection should be for one fault at a time, in a single branch, so that its effect can be studied and compared with the fault-free circuit, as well as with the other faults.

4.1.6 Failure Mode and Effect Analysis (FMEA)

In the context of the analysis and injection of mechanical failures, one cannot omit the FMEA, which is often a core task in reliability engineering, safety engineering, and quality engineering [152]. In fact, this methodology is used to analyze the failure or defect modes of a process, product, or system, analyze its causes and evaluate what the effects are on the entire system. Generally, the analysis is carried out in advance and is therefore based on theoretical and not experimental considerations. The process consists of reviewing as many components, assemblies, and subsystems as possible to identify potential failure modes in a system and their causes and effects. For each component, the failure modes and their resulting effects on the rest of the system are recorded in a specific FMEA worksheet. A successful FMEA activity helps identify potential failure modes based on experience with similar products and processes or based on common physics of failure logic. It is widely used in development and manufacturing industries in various phases of the product life cycle. Effects analysis refers to studying the consequences of those failures on different system levels [153, 154]. So, in detail, the procedure always acts on the device under test in optimal conditions:

- one fault at a time;
- the component is powered normally and receives correct commands;
- all consumable resources are present in sufficient quantity.

The analysis starts by listing the functions that the design needs to fulfill since a design is meant to be the only one possible solution to perform functions that need to be satisfied. Thus, faults are studied as variations of these functions that describe the system, through some parameters: the probability of the fault occurring, the severity of the issue, and whether and how the failure can be detected. These parameters identify the risk level of the problem, which can range from low to unacceptable. Other useful variables are brought back from the analysis on a single fault, that is the potential causes, the effect of the failure at low and system levels, and how to repair it.

However, FMEA has not yet been automated, as no software tool is used to apply this methodology. The use of neural network techniques to cluster and visualize failure modes were suggested, but nothing more. Furthermore, the FMEA worksheet is hard to produce, hard to understand and read, as well as hard to maintain.

4.1.7 HIFSuite framework

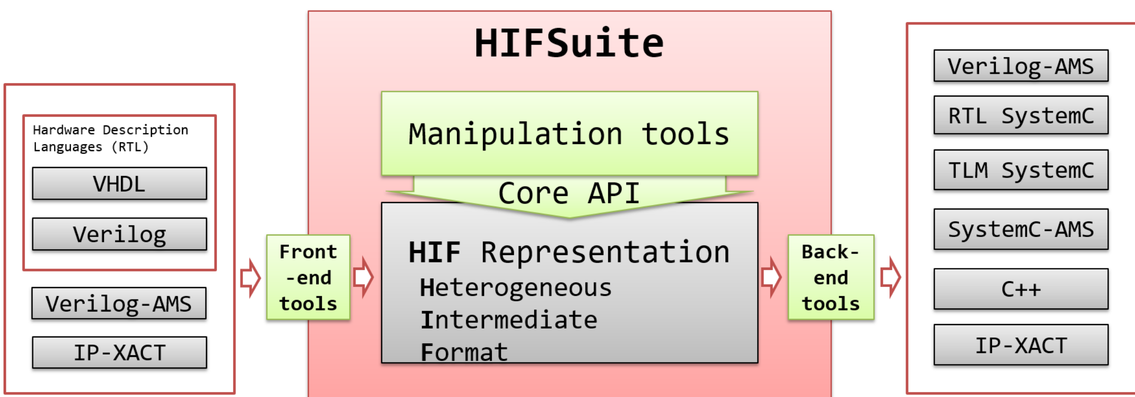


Fig. 4.3: HIFSuite overview

HIFSuite [155] is a set of tools and application programming interfaces (APIs) that provide support for the modeling of HW/SW systems. Its core is the *HDL Intermediate Format* which is a proprietary language used by all the tools. One of the functionalities of this framework is the manipulation of HDL designs to obtain their functional equivalent C++ versions. Another feature is the possibility to translate a design from one HDL language to another. The HIFSuite framework contains three different kinds of tools (Figure 4.3):

- `front-end tools` parse HDL descriptions to generate the corresponding HIF representations;
- `back-end tools` parse HIF representations and converts them into HDL or pure C++ descriptions;
- `manipulation tools` manipulate the HIF representation introducing faults, abstracting data types, abstracting processes, *etc*.

Some front-end tools are *vhdl2hif* and *verilog2hif*, while some back-end tools are *hif2vhdl*, *hif2sc* and *hif2vp*. The *hif2sc* tool can generate both SystemC TLM and C++ code from a given HIF description. The *hif2vp* can automatically produce an *FMI compliant XML description* or an *Open Virtual Platform (OVP)* format. Some manipulation tools are:

- `ddt`: it converts the HDL datatypes of a description to C++ pure datatypes which are more efficient during simulation;
- `a2tool` [156]: it abstracts an *register transfer level* (RTL) description to its *transactional* (TLM) semantic equivalent one;
- `analyst` [157]: it generates a discrete event model starting from an analog design (taken from *verilog AMS* or *VHDL-AMS*);
- `composer`: it can merge multiple HIF descriptions into a single description by specifying an external interface and how to interconnect the sub-modules [158].

4.2 Represent a mechanical model with its equivalent electrical model

Over the years, electrical equivalent models have been used in many industrial and research fields for representing the functionality of complex systems. Electrical circuits have been used to represent systems physically different, e.g., the variability of the characteristic parameters of automotive batteries [159, 160], internal combustion engine [161], acoustic micro-speakers [162], Surface Acoustic Wave (SAW) filter [163], Radio Frequency (RF) Micro-Electro-Mechanical System (MEMS) disk resonators [164], a human cochlea [165], *etc*.

Through an equivalent electrical model, it is possible to approximate the behavior of a dynamic system even when it is non-linear. For example, to represent a rotational/translational mechanical system with its electrical equivalence it is necessary to use analogies. Through analogies, it is in fact possible to link the physical quantities of a domain to another physical domain. The main analogies defined between the mechanical and the electrical domain are the impedence [166] and the mobility analogy [167, 168]. The mathematical behavior of the simulated electrical system is identical to the mathematical behavior of the represented mechanical system. Once a system is represented with its electrical equivalent, it is possible to simulate it with a modern simulator (e.g., Spectre and Eldo), allowing for accurate and fast simulations. Alternatively, it is possible to simulate these models through multi-physics simulators (e.g., Matlab Simulink/Simscape, Siemens AMEsim). Through an HDL simulator, it is possible to combine the equivalent electrical model with accurate digital models, without doing co-simulation using standards like Functional Mock-up Interface [169].

These types of analogies become even more appreciable when analyzing an electro-mechanical system. An example of these heterogeneous systems is the Direct Current (DC) Motor, where both electrical and mechanical domains are present. The electrical part controls the motor in voltage and through an Electromotive Force (EMF) we are able to move the mechanical part of the motor, the shaft. In this case, representing the mechanical part with its electrical equivalent we can simulate the entire DC motor represented as an electrical circuit. It is possible to select based on the knowledge acquired when to use the force-voltage analogy or the force-current analogy, as equivalent results can be obtained through one or the other. Figure 4.4 shows two examples of analogies between mechanical and electrical domains.

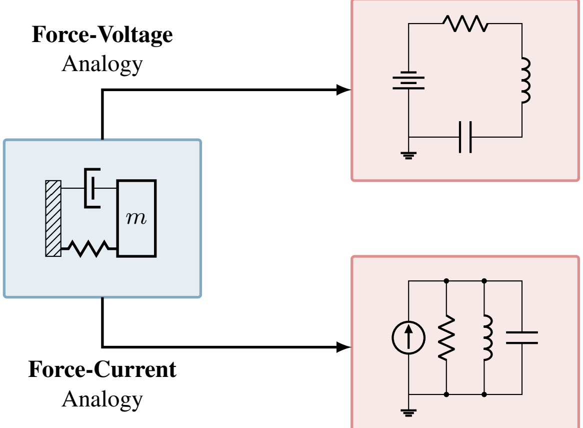


Fig. 4.4: Analogies between mechanical (left) and electrical (right) system.

The *force-voltage* analogy is called direct analogy and is the most intuitive [170], but it requires passing through an intermediate representation (mechanical network) to go from the mechanical domain to the electrical domain. While the *force-current* analogy is more complex but allows to preserve the topology of the mechanical model. Section 4.2.3 and 4.2.2 describes these specific analogies in details. These two

analogies are reciprocal, *force-voltage* analogy associate voltage to force and current with velocity, while *force-current* analogy associate voltage with velocity and current with force. From a mathematical point of view, neither analogy is superior to the other, because when this analogy exists it can lead to equivalent descriptions that remain valid and consistent in both cases [171]. The choice remains arbitrary because through these analogies it is possible to obtain dual results. The impedance analogy preserves the analogy between electrical impedance and mechanical impedance whereas the mobility analogy does not. On the other hand, the mobility analogy preserves the topology of the mechanical system when transferred to the electrical domain whereas the impedance analogy does not. However, all laws of circuit analysis, such as Kirchhoff's circuit laws, that apply in the electrical domain also apply to both these analogies.

4.2.1 Mechanical Networks

When the *force-voltage* analogy is applied, it is necessary to pass through a representation of the dual mechanical network to derive the electrical equivalent model. This intermediate step is required because using this analogy does not preserve the topology of the mechanical network in the construction of the electrical circuit. This intermediate mechanical model is called a mechanical network: it is an abstract interconnection of the mechanical elements that compose the system via nodes.

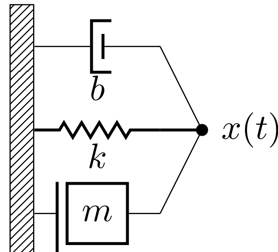


Fig. 4.5: Mass-Spring-Damper mechanical network.

In Figure 4.5, the node x is present to connect the different mechanical components that model the Mass-Spring-Damper (Figure 4.6). The node x represents exactly the displacement ($x(t)$ of the mass in the mechanical system. Then, if the components are connected to the same displacement x_n then they will be connected in series in the electric circuit since they are affected by the same electric current (*force-voltage* analogy consequence). While if the components are connected at different displacements, then they will be connected in parallel in the electrical circuit since they are not affected by the same electric current. By using this intermediate description is easy to derive the state space model of our system. In the same way, when we represent a system by using a Bond graph, it is easy to derive the state space model of the original system [168] because the same information stored inside a mechanical network is stored inside a Bond graph.

4.2.2 Impedance Analogy

Through the *force-voltage* analogy, a mechanical system can be represented as an electrical one. Each mechanical component has its corresponding electrical one, as shown in Table 4.4. The mapping associates resistance with damping, the inductance with a mass, the capacitance with a spring, *etc*. The *effort* variable, force in the mechanical domain, is represented by voltage in the electrical domain; instead, the *flow* variable, velocity in the mechanical, becomes current in the electrical domain. The other equivalence relations shown are obtained by mathematical equality between the two domains: for example, a damper is equivalent to a

resistor since both represent energy loss in their own domain. Due to the mathematical analogy between these quantities, it is possible to realize different mechanical systems.

Table 4.4: Relationship between mechanical and electrical variables in the force-voltage (impedance) analogy.

Type	Mechanical translation	Mechanical rotation	Electrical
Effort variable	Force	Torque	Voltage
Flow variable	Velocity	Angular velocity	Current
Elements	Damping	Rotational resistance	Resistance
	Mass	Moment of inertia	Inductance
	Compliance	Rotational compliance	Capacitance
	Mechanical impedance	Mechanical impedance	Electrical impedance

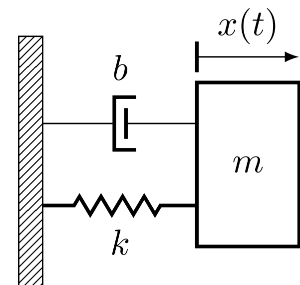


Fig. 4.6: Mass-Spring-Damper schema.

Mass-Spring-Damper system

Let us consider as an example the Mass-Spring-Damper system, shown in Figure 4.6. This model is well known in the mechanical world, as it is an oscillator used for the realization of many more complex systems. There are many works that refer to this system to study different but equivalent ones [172, 173, 174]. This system consists of a mass (m) connected to the ground reference by a spring (k) and a damper (b). The behavior of the Mass-Spring-Damper is modeled through the differential Equation 4.1, where the force (F) applied to the system is given by the sum of the equations of the single components. In the real system, force is applied to the mass, causing it to oscillate due to spring and damper.

$$F(t) = m\frac{d\dot{x}}{dt} + b\dot{x} + k\int \dot{x}dt \tag{4.1}$$

Through the *force-voltage* analogy, it is possible to describe this mechanical system as a simple electrical system composed of a resistance, an inductance, and a capacitance: the RLC circuit, shown in Figure 4.7. The behavior of the RLC is defined through the differential Equation 4.2.

$$V(t) = L\frac{di}{dt} + Ri + \frac{1}{C}\int idt \tag{4.2}$$

Note that the equation and also the behavior of the two systems are totally equal: they differ only for the name of the physical quantities because of the one-to-one mapping with analogous physical quantities between the two domains, shown in Table 4.4. The Listing 4.1 shows this circuit implemented in the Verilog-

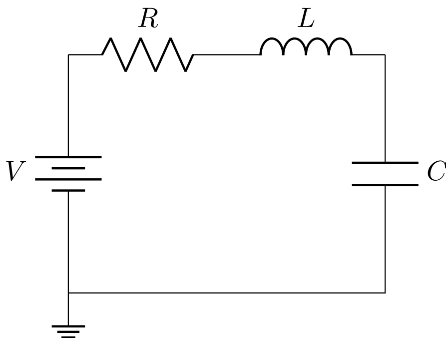


Fig. 4.7: The RLC circuit, the electrical equivalent of the Mass-Spring-Damper system according to impedance analogy.

AMS environment.

The main advantage of the impedance analogy over its alternative, the mobility analogy (Section 4.2.3), is that it maintains the analogy between electrical and mechanical impedance. Consequently, a mechanical impedance is represented as an electrical impedance, and a mechanical resistance is represented as an electrical resistance in the electrical equivalent circuit. It is also natural to think of force as analogous to voltage (generator voltages are often called electromotive force) and velocity as analogous to current. It is this basic analogy that leads to the analogy between electrical and mechanical impedance.

However, the principal disadvantage of the impedance analogy is that it does not preserve the topology of the mechanical system. Elements that are in series in the mechanical system are in parallel in the electrical equivalent circuit and vice versa. This is why passing through mechanical networks helps to obtain the electrical equivalent circuit. So, in order to convert the mechanical model (Figure 4.6) to the electrical equivalent (Figure 4.7), it is necessary to make an intermediate step represented in Figure 4.5 which represents a mechanical network [170, 175].

```
1  `include "disciplines.vams"
2  `include "constants.vams"
3  `timescale 1us / 1us
4
5  module rlc (t1, t4);
6      // PARAMETERS ----------------------
7      parameter real r = 7;
8      parameter real l = 0.15;
9      parameter real c = 100e-6;
10
11     // PORTS ---------------------------
12     input t1, t4;
13
14     // NODES ---------------------------
15     electrical t1, t2, t3, t4;
16
17     // BRANCHES ------------------------
18     branch(t1,t2) res;
19     branch(t2,t3) ind;
20     branch(t3,t4) cap;
21
22     // BEHAVIOR ------------------------
23     analog begin
24
25         I(res) <+ V(res)/r;
26         I(ind) <+ idt(V(ind))/l;
27         I(cap) <+ ddt(V(cap))*c;
28     end
29 endmodule
```

Listing 4.1: RLC circuit modeled in Verilog-AMS.

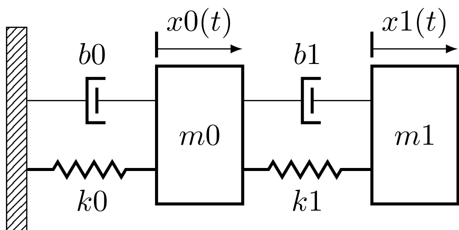


Fig. 4.8: Tuned Mass-Spring-Damper schema.

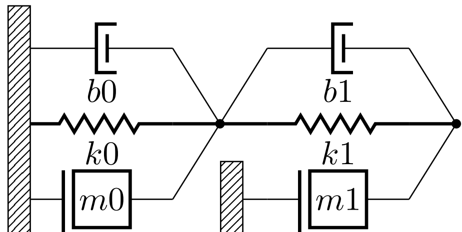


Fig. 4.9: Tuned Mass-Spring-Damper mechanical network.

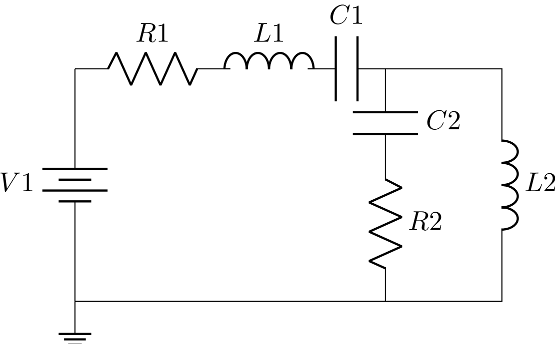


Fig. 4.10: Electrical equivalent of the tuned Mass-Spring-Damper system (double-RLC).

Tuned Mass-Spring-Damper system

Let us consider the tuned Mass-Spring-Damper system, shown in Figure 4.8. This system consists of a mass (`m0`) connected to the ground reference by a spring (`k0`) and a damper (`b0`), that is connected in turn to a second mass (`m1`) through a spring (`k1`) and a damper (`b1`). Generally, the second mass `m1` is smaller with respect to the first mass `m0`. This particular configuration of two Mass-Spring-Damper allows building a system for damping the amplitude in one oscillator by coupling it to a second oscillator. If tuned properly the maximum amplitude of the first oscillator in response to a periodic drive will be lowered. The behavior of the tuned Mass-Spring-Damper is modeled through the Equation (4.3) and Equation (4.4).

$$m_1\ddot{x_1} = F + k_2(x_2 - x_1) + b_2(\dot{x_2} - \dot{x_1} - k_1 x_1 - b1\dot{x_1}) \tag{4.3}$$

$$m_2\ddot{x_2} = -k_2(x_2 - x_1) - b_2(\dot{x_2} - \dot{x_1}) \tag{4.4}$$

Through the *force-voltage* analogy, it is possible to describe this mechanical system as a simple electrical system composed of two resistance-inductance-capacitance (RLC) circuits. Figure 4.10 is showed the electrical equivalent of the tuned Mass-Spring-Damper (Figure 4.8) built trough the *force-voltage* analogy named double-RLC to make the reference to this circuit easier. The behavior of the double-RLC is defined through the Equation (4.5) and Equation (4.6).

$$V_1 = V_{in} + \frac{1}{C_2}\int (i_2 - i_1) + R_2(i_2 - I_1) - \frac{1}{C_1}\int i_1 - R_1 i_1 \tag{4.5}$$

$$V_2 = -\frac{1}{C_2}\int (i_2 - i_1) - R_2(i_2 - i_1) \tag{4.6}$$

In order to convert the mechanical model (Figure 4.8) to the electrical equivalent (Figure 4.10), it is necessary to make an intermediate step represented in Figure 4.9 which represents a mechanical network [176, 175]. When the *force-voltage* analogy is used, it is necessary to pass through a representation of the dual mechanical network to derive the electrical equivalent model. This intermediate step is required because using this analogy does not preserve the topology of the mechanical network in the construction of the electrical circuit. In Figure 4.9, two nodes $x1$ and $x2$ are present to connect the different mechanical components that model the tuned Mass-Spring-Damper. These two nodes $x1$ and $x2$ represent exactly the displacements ($x0(t)$ and $x1(t)$) of the two masses in the mechanical system. Then, if the components are connected to the same displacement x_n then they will be connected in series in the electric circuit since they are affected by the same electric current (*force-voltage* analogy consequence). While if the components are connected at different displacements, then they will be connected in parallel in the electrical circuit since they are not affected by the same electric current. By using this intermediate description is easy to derive the state space model of our system. In the same way, when we represent a system by using a Bond graph, it is easy to derive the state space model of the original system [168] because the same information stored inside a mechanical network is stored inside a Bond graph. The Listing 4.2 presents the implementation of the double-RLC (electrical circuit shown in Figure 4.10) modeled in Verilog-AMS.

```
1  `include "disciplines.vams"
2  `include "constants.vams"
3  `timescale 1us / 1us
4
5  module double_rlc (t1, t6);
6      // PARAMETERS ----------------------
7      parameter real r0 = 7;
8      parameter real l0 = 0.15;
9      parameter real c0 = 100e-6;
10     parameter real r1 = 5;
11     parameter real l1 = 0.10;
12     parameter real c1 = 50e-6;
13
14     // PORTS ---------------------------
15     inout t1, t6;
16
17     // NODES ---------------------------
18     electrical t1, t2, t3 , t4, t5, t6;
19
20     // BRANCHES ------------------------
21     branch(t1,t2) res0;
22     branch(t2,t3) ind0;
23     branch(t3,t4) cap0;
24     branch(t4,t5) cap1;
25     branch(t5,t6) res1;
26     branch(t4,t6) ind1;
27
28     // BEHAVIOR ------------------------
29     analog begin
30         V(res0) <+ I(res0) * r0;
31         V(ind0) <+ l0*ddt(I(ind0));
32         V(cap0) <+ idt(I(cap0))/c0;
33
34         V(cap1) <+ idt(I(cap1))/c1;
35         V(res1) <+ I(res1) * r1;
36         V(ind1) <+ l1*ddt(I(ind1));
37     end
38 endmodule
```

Listing 4.2: Double-RLC circuit modeled in Verilog-AMS.

4.2.3 Mobility Analogy

Similar to the *force-voltage* analogy, a mechanical system can be represented as an electrical one through the *force-current* analogy. Each mechanical component has its corresponding electrical one, as shown in Table 4.5. The main difference lies in the association between effort and flow variables, which are dual to the impedance analogy. The mapping associates resistance with damping, the capacitance with a mass, the inductance with a spring, *etc*. The *effort* variable, force in the mechanical domain, is represented by the current in the electrical domain; instead, the *flow* variable, velocity in the mechanical, becomes voltage in the electrical domain. The other equivalence relations shown are obtained by mathematical equality between the two domains: for example, a damper is equivalent to a resistor since both represent energy loss in their own domain. Due to the mathematical analogy between these quantities, it is possible to realize different mechanical systems.

Table 4.5: Relationship between mechanical and electrical variables in the force-current (mobility) analogy.

Type	Mechanical translation	Mechanical rotation	Electrical
Effort variable	Force	Torque	Current
Flow variable	Velocity	Angular velocity	Voltage
Elements	Damping	Rotational resistance	Admittance
	Mass	Moment of inertia	Capacitance
	Compliance	Rotational compliance	Inductance
	Mechanical impedance	Mechanical impedance	Electrical impedance

Let us consider the Mass-Spring-Damper system again, Figure 4.6: using this analogy rather than force-voltage, different differential equations, and electrical circuits are obtained. The behavior of the mechanical system is already described by differential Equation 4.1, but the electrical equivalent, shown in Equation (4.7), is different from the Equation (4.2) due to the *force-current* analogy.

$$i(t) = C\frac{dv}{dt} + \frac{v}{R} + \frac{1}{C}\int vdt \tag{4.7}$$

Through the *force-current* analogy, it is possible to describe this mechanical system as a simple electrical system composed of resistance, inductance, and capacitance. However, it differs from the RLC circuit because the components are in parallel instead of being connected in series. The circuit is named parallel RLC, and it is shown in Figure 4.11. Although the equation and topology of the circuit are different from that of

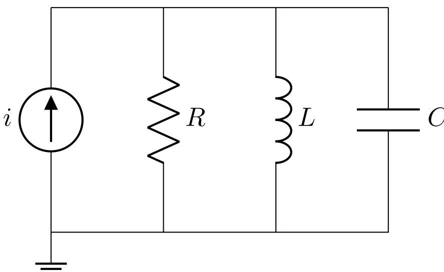


Fig. 4.11: The parallel RLC circuit, electrical equivalent of the Mass-Spring-Damper system according to mobility analogy.

the Mass-Spring-Damper system and, thus, the RLC, the behavior is the same as the other two systems. The principal advantage of the mobility analogy over its alternative, the impedance analogy, is that it preserves

the topology of the mechanical system. Elements that are in series in the mechanical system are in series in the electrical equivalent circuit, and elements in parallel in the mechanical system remain in parallel in the electrical equivalent. However, the principal disadvantage of the mobility analogy is that it does not maintain the analogy between electrical and mechanical impedance. Mechanical impedance is represented as an electrical admittance, and mechanical resistance is represented as an electrical conductance in the electrical equivalent circuit. Force is not analogous to voltage (generator voltages are often called electromotive force), but rather, it is analogous to current.

4.3 Inferring mechanical fault models from the electrical domain

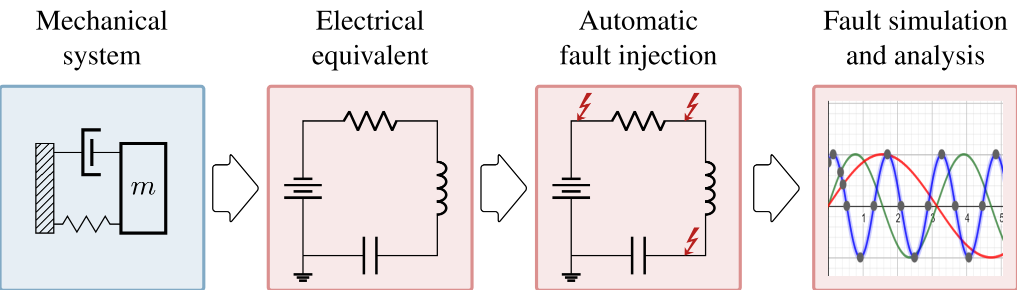


Fig. 4.12: Flow of proposed methodology to derive mechanical fault modes. It starts from a mechanical system, which is transformed into an electrical equivalent one, then faults compliant with the *IEEE P2427* standard are injected and simulated. The results are used to build a taxonomy of mechanical faults implied by the equivalent electrical model.

The proposed flow to derive mechanical fault modes is shown in Figure 4.12. The first step is to identify the electrical circuit corresponding to the mechanical system under consideration. However, the analogy alone is not enough to accomplish this conversion: it only offers an equality relation between variables of physical domains, while the topology of the circuit is not specified. Then, it is necessary to describe the mechanical model as a mechanical network: it gives an idea of how the flow and effort variables act in the system. Due to this intermediate operation, it is easier to create the electric circuit, and it allows the automation of the process. Once the mechanical system equivalent circuit is obtained from the mechanical network, it is implemented within the Verilog-AMS environment. Verilog-AMS is a hardware description language that can model both analog and digital systems by describing differential equations of different physical domains. It can also perform transient and frequency simulations of electrical, mechanical, magnetic, and thermal systems. In addition, controllers that interface analog and digital parts of the model can be built with this language. There are several equivalent hardware description languages, for example, VHDL-AMS, but Verilog-AMS was chosen for working convenience.

At this point, it is possible to simulate the electrical circuit and study its behavior before injecting faults. When the devices are working, the faults are injected into the model: and the analysis of the system behavior in relation to the mechanical domain is performed. Specifically, the behaviors of the faulty electrical circuit were interpreted in the mechanical domain: if an electrical fault has significance and is replicated in other mechanical systems, then the electrical fault pattern is included in the classification of the mechanical fault. However, some behaviors due to electrical faults do not make sense in the mechanical world, so they have not been classified. The proposed classification has been tested on various mechanical systems in the Verilog-AMS environment. A consideration of other physical domains is proposed: the electrical domain is also subject to other analogies, for example, with the magnetic, thermal, and hydraulic domains. A study

similar to the previous one is possible, looking for analogies between electrical faults and those of the domain in the object. The electrical domain remains the chosen one for this methodology, given the breadth of available fault models and to be able to apply automatic fault injection.

4.3.1 Fault Modeling in Verilog-AMS

Defining generic fault models that can have meaning in any physical discipline is difficult, if not impossible. Even if Verilog-AMS supports different disciplines, specific physical phenomena affect each discipline in different ways. Faults are injected in the Verilog-AMS description by adding the corresponding equations to add components or relationships in parallel or series. This is allowed in Verilog-AMS with few restrictions. Verilog-AMS allows defining parallel components between the same pair of nodes only if both have on the left-hand-side the *flow* between a couple of nodes, and it allows placing components in series between the same pair of nodes only if both have on the left-hand-side the *potential* between a couple of nodes. Mixing equations that have potential and flow on the left-hand side between the same node is allowed by Verilog-AMS. However, in this case, the Language Reference Manual states that the value retention rule is applies [117]. According to this rule, when a contribution of a specific nature is defined on a branch of the circuit, introducing a new contribution of the same nature will result in a summation of the effects, while a new contribution of the opposite nature will discard the previous contribution. For instance, if we write the voltage contribution between two nodes and afterward, we define a current contribution between the same two nodes, the voltage one is discarded. To add the current contribution is thus necessary to find an alternative solution, e.g., by adding a new circuit node. However, these measures, described in depth in [177, 178], are easier to handle if an alias is declared for each branch.

Let us now see how electrical faults, reported in Section 4.1.3, are injected into a Verilog-AMS description. In the *electrical* discipline, both open, short, voltage, and current sources faults are implemented as *saboteurs*. Open and short circuit failures are realized by adding a resistor to the description. Consider the following equation as an example:

```
I(p, n) <+ ddt(V(p, n)) * c;
```

To inject an open fault, a new node `p_n` and the following statement have to be added to the previous one:

```
I(p, p_n) <+ ddt(V(p, p_n)) * c;
I(p_n, n) <+ V(p_n, n) / r;
```

In an open circuit, the *r* value is extremely high, thus modeling a wiring break that causes the current to cease (even if the electric field will still be present). Since the open fault is injected in series, a new node would not have been necessary if the equation had been defined in terms of the potential variable.

In order to inject a short fault in the previous example, we need to add the following statement:

```
I(p, n) <+ ddt(V(p, n)) * c;
I(p, n) <+ V(p, n) / r;
```

In the case of a short circuit, the *r* value becomes extremely low (a nearly zero ohms path), resulting in a large current flow and a potential difference between the pins of the new path equal to zero. This fault is injected in parallel, so there is no need for an intermediate node since the equation is described in terms of the flow variable.

Regarding the injection of voltage and current sources, it is necessary to add a contribution of the desired magnitude in the form of a *pulse*. Considering the previous example again, a voltage source is injected as follows:

```
I(p, p_n) <+ ddt(V(p, p_n)) * c;
V(p_n, n) <+ Pulse;
```

Similarly, a current source can be injected into the circuit:

```
I(p, p) <+ ddt(V(p, p_n)) * c;
I(p, n) <+ Pulse;
```

Finally, parametric faults can be injected as alterations of the parameters defined during model design: no new dependency is added to the model, and such faults are *mutants*.

4.3.2 Exemplification on the double-RLC circuit

Consider now the injection of electrical fault models (Section 4.1.3) injected into this model and compared with fault-free behavior. In addition, for each fault, the equivalent mechanical behavior is studied. The normal response of the system to an incoming step wave is shown in Figure 4.13. The input signal represents a force applied on the mass $m0$, equivalent to the electrical voltage according to the analogy. As we can see from the graph, the smaller mass ($m1$) oscillates less than, the larger one ($m0$). This is due to the structure of the system, which is also mainly used in seismic contexts, where the mass $m0$ is a building while $m1$ is a balancing system in case of an earthquake. Let's start with the open failure: an open fault injected into the resistor branch $r0$, as shown in Figure 4.14. In particular, a new branch is injected in series to the present one with a very high-value resistor to simulate a break in the original branch itself. This is how the fault is injected into the code:

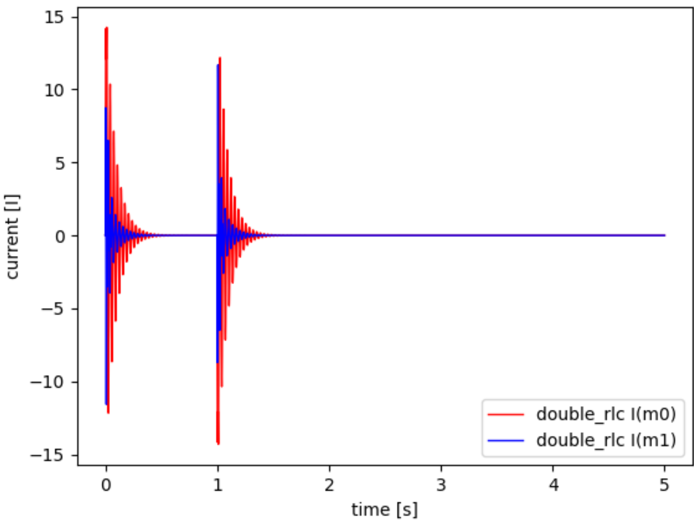
```
...
branch(t1, temp) res1;
branch(temp, t2) open_fault;
...
I(open_fault) <+ V(open_fault) / 1e09;
...
```

Figure 4.14 clearly shows how the injected resistor completely drops the current swing in the first part of the circuit. In the mechanical domain, this means that the velocity of motion of mass $m0$ is reduced simply to the displacement due to the input force. Note how instead, the mass $m1$ oscillates anyway in response to the exerted force since it is without additional brakes. The mechanical failure identified is, therefore, an increase in the resistance to movement (which may be due to an additional interfering brake or an increase in the friction of the surface on which the component is moving) of the component from the system. This fault has the same effect even when injected onto the other components.

Moving now to the short fault, a new branch is simply injected in parallel, in this case, again to the branch of $r0$. Here is how the fault is injected into the code:

```
...
branch(t1, t2) short_fault;
...
I(short_fault) <+ V(short_fault) / 1;
...
```

From the outcome of the system, shown in Figure 4.15, we can see how the current oscillation has increased out of proportion. This means, in the mechanical domain, total isolation of the damper $b0$ (corresponding to the resistance $r0$). The mechanical failure identified then is a disconnection of the component from the system. This fault has the same effect even if injected on the capacitor, then on the spring (in that case, the system would be damped above normal).



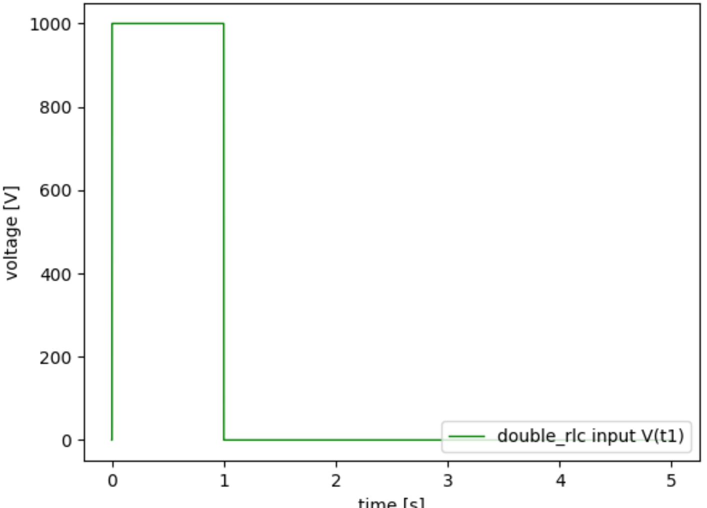


Fig. 4.13: Normal response of the Double-RLC fault-free.

Let us now look at the source faults: the voltage source is a fault that must be inserted in series to an existing branch, such as the open. In particular, an additional sine wave is inserted:

```
...
branch(t2, temp) ind1;
branch(temp, t3) voltage_fault;
...
I(voltage_fault) <+ 1e03 * sin(`M_PI * 2 * $abstime);
...
```

As we see in Figure 4.16, the current oscillations are affected by the input signal in the circuit. This fault has been injected in series with the branch representing the inductance $l0$, so the ground $m0$. This, on a mechanical level, indicates an abnormal force entering the system, in this case, on mass $m0$. The whole

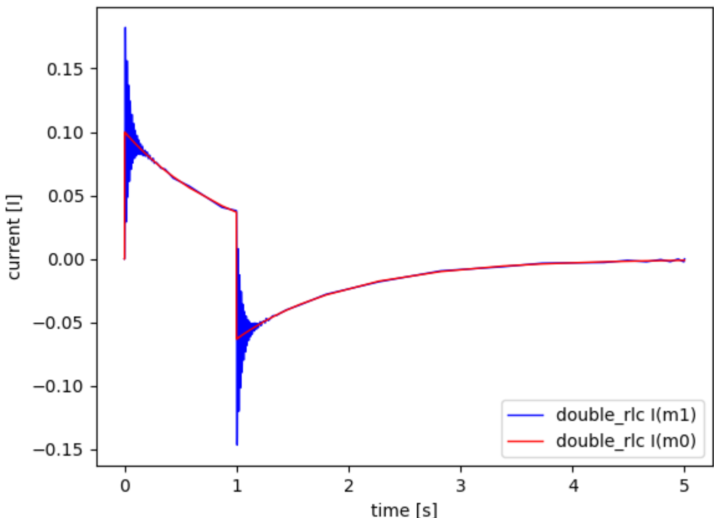


Fig. 4.14: Response of the faulty Double-RLC, open.

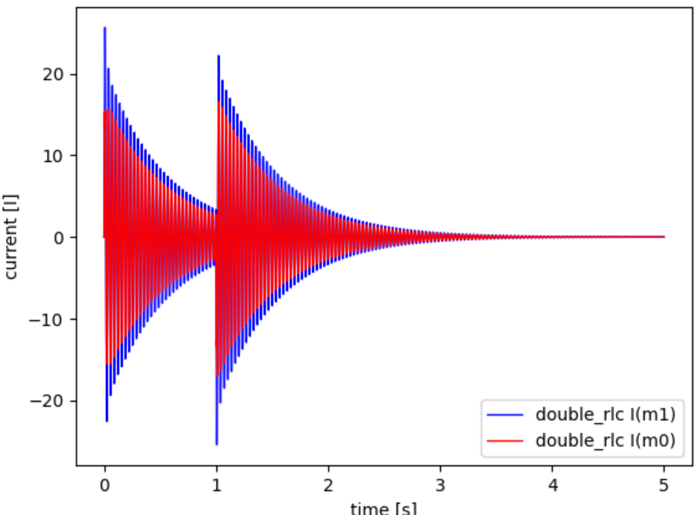


Fig. 4.15: Response of the faulty Double-RLC, short.

system is affected by this anomaly, and its displacement is modified by the injected force. If the fault had been inserted in series with any other branch, the result would have been the same.

Finally, let us consider the current source: unlike the voltage source, here, an increase in current is injected in parallel to the branch $l0$, i.e. $m0$. There is a fault injection in the circuit code:

```
...
branch(t2, t3) current_fault;
...
I(current_fault) <+ (1e02 * $abstime);
...
```

The system outcome is shown in Figure 4.17. The system is governed by the current that is injected into the system: it is important to note that, at the mechanical level, the system oscillates much less at the points

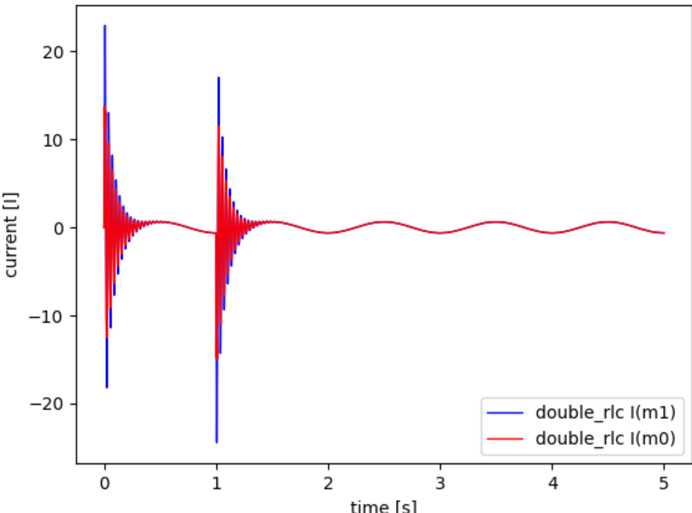


Fig. 4.16: Response of the faulty Double-RLC, voltage source.

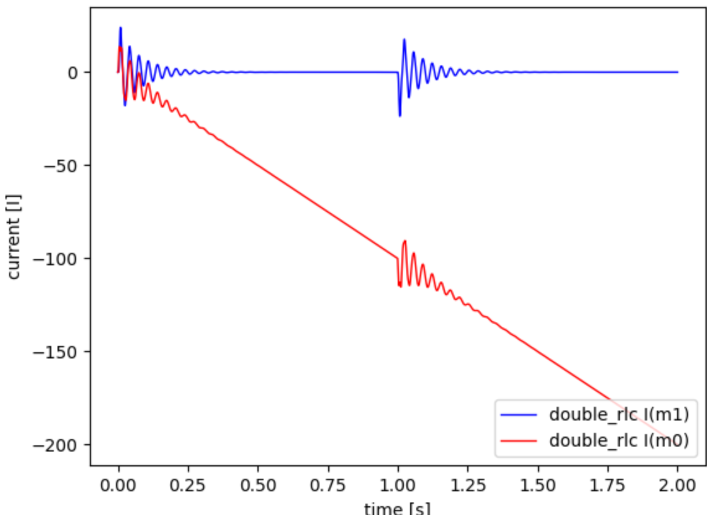


Fig. 4.17: Response of the faulty Double-RLC, current source.

where the force (input voltage) is applied. Instead, the mass $m0$ follows, unnaturally, the direction imposed by the input flow variable. The mass $m1$ instead follows its normal trajectory. For parametric faults, the outcome in Figure 4.14 is the same as if we increased the value of the resistor r0 without adding an open. Similarly, by decreasing the value of r0, we would have an outcome similar to the short fault (Figure 4.15). The approach described so far is the procedure followed for the analysis and mechanical faults in relation to the electrical domain. In particular, we have shown that, by injecting electrical fault models onto a circuit equivalent to a mechanical system, it is possible to inject the mechanical faults identified by the proposed mechanical fault taxonomy Section 4.3.3.

Table 4.6: Mechanical fault taxonomy inferred from the analog fault injection in the equivalent model.

Mechanical Behavior	Mechanical Effect	Mechanical Fault	Electrical Fault
Brake/ friction	Added/increased braking force on failed component	Galling/Seizure, Creep, Spalling, Wear/Corrosion	Open
Disconnection	The failed component detaches from the system	Rupture, excess of Backlash	Short
External force	An abnormal (external) force affects the component	It can cause deformations or cracks or ruptures (from impact of fatigue)	Voltage source
Limited movement	The direction of displacement of a component is abnormally modified	Rupture, Deformation, Wear	Current source
Parametric	Intrinsic characteristics of the failed component altered	Wear (component aging) and all the parameters changes	Parametric

4.3.3 A new mechanical fault taxonomy

The idea of working with the electrical equivalent and not directly with a mechanical model stems from the fact that mechanical faults are difficult to inject into mechanical systems [179]. Through the injection of analog faults within the electrical equivalent, it is possible to derive mechanical fault classes for given mechanical components.

However, physical analogies do not provide any information about failures: it is not always granted that electrical fault models have correspondence in the mechanical world. In fact, behavior is mathematically equal across domains, but that does not mean it has to be functionally equal. Because of that, in order to define a possible mapping between electrical and mechanical faults, there is a need for testing and comparing their behavior. In particular, it is necessary to simulate the equivalent faulty electrical circuit using electrical fault injection techniques and to study the obtained behavior from the mechanical point of view. If the behavior has a meaning from both an electrical and mechanical point of view, then there is a correlation between faults belonging to different domains.

The *open circuit* fault represents an arbitrarily valued resistor injected into a line of the circuit. Ideally, with a very high value, the fault simulates a line break. For example, the electrical open fault is equivalent to a mechanical braking agent because both, when injected, significantly reduce the flow variable, i.e., velocity and electric current. We can think of this failure as an increased coefficient of friction on the motion surface due to temperature, wear, or the presence of debris. Similarly, the other electrical faults at the behavioral level have been applied: shorts and voltage, and current sources. The *short circuit* fault consists of an unwanted branch between two points of the circuit that originally was not supposed to touch each other. This fault can be assimilated to a disconnection of certain mechanical components from the rest of the system, such as springs or dampers. The separation of a component from the rest of the system could happen if it is affected by excessive backlash or rupture. Voltage and current *sources* represent unwanted changes in voltage and current at a given point in the circuit. These changes might be caused by interference coming from surrounding electrical circuits or by alpha particles coming from outer space hitting a portion

of the circuit. According to the analogy, force is equivalent to voltage: therefore, a voltage source can be seen as an unexpected external force on a certain component. Similarly, velocity is equivalent to electric current: therefore, a current source is a variation of the displacement velocity of the component in which it is injected. As for *parametric faults*, they are equivalent in both domains. In Table 4.3, there are some mechanical failures that can be assimilated to the faulty behaviors obtained through the simulation [146, 151].

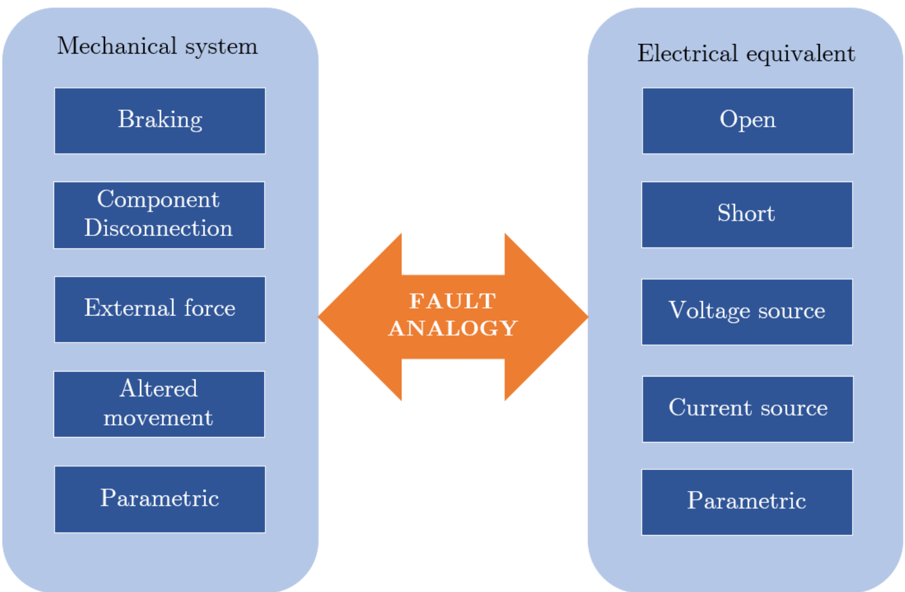


Fig. 4.18: Fault analogy described by the mechanical taxonomy.

As shown in Figure 4.18, the*force-voltage* analogy also extends to the failure models in the respective physical domains. The proposed taxonomy was drafted according also to the mechanical systems analyzed. By studying more complex systems with different components, the taxonomy could be enriched with new data.

The main contribution of this section is a new mechanical fault taxonomy presented in Table 4.6. The table shows an extension of the analogy: in particular, the injected electrical fault models and the detected equivalent mechanical behaviors are shown. The taxonomy has been derived mainly from the simulation of several mechanical systems through electrical analogy: e.g., a mass-spring-damper, a tuned mass-spring-damper, a double pendulum, and a DC motor. The proposed taxonomy has been built to show how mechanical fault behaviors can be derived by simulating faulty electrical equivalent systems. Thus, this fault taxonomy could be extended by simulating more complex mechanical systems. The next section will illustrate the procedure of multi-domain fault injection and fault simulation exemplified on a complex system: a DC motor with a gear train.

4.4 Multi-domain fault modeling

This section proposes an application of the multi-domain fault injection applied to a complex system: a DC motor with a gear train shown in Figure 4.20. In detail, the electrical faults are applied to the electrical part of the system, and the mechanical faults (derived in Section 4.3) are applied directly to the mechanical part. The DC motor with a gear train is composed of two main components: the DC Motor and the gear train.

4.4.1 DC Motor

The direct current (DC) motor is an electromechanical system that converts electrical energy into mechanical energy through the interaction between the magnetic field of the motor and an electric current in a wire winding [180].

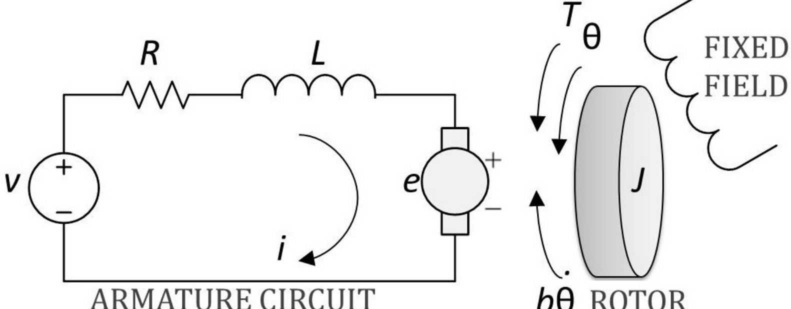


Fig. 4.19: DC motor represented as an equivalent circuit of the armature windings and the free-body diagram of the rotor.

Equation-based model of the DC motor

The DC motor can be modeled with an equivalent circuit of the armature and the free-body diagram of the rotor. A schematic of the motor is shown in Figure 4.19. The following equations model the internal dynamic of the motor [181]:

$$v = K_M \cdot \omega + R \cdot i + L \cdot di/dt \tag{4.8}$$

$$\tau = K_T \cdot i - d \cdot \omega - j \cdot d\omega/dt \tag{4.9}$$

where v is the input voltage source applied to the motor's windings to control the velocity of the rotor; variable i represents the current through the windings; variable ω is the angular velocity of the shaft; the variable τ is the torque on the shaft. The motor coefficients K_M and K_T are used to dimension the size of the motor: coefficient K_M is used to compare motors and to calculate temperature rise based on the dissipated power, while coefficient K_T is used to calculate the required current based on the required torque. The rotor and the shaft are assumed to be rigid, and the friction torque is proportional to the shaft angular velocity.

DC Motor modeled in Verilog-AMS

Modeling the DC motor with the Verilog-AMS language requires the use of the two disciplines discussed in this work: the *electrical* discipline and the *rotational_omega* discipline. The electrical circuit of the DC motor (equation 4.8) is modeled through an electrical network made of three components: a voltage source (`Vm`), a resistor (`R`), and an inductor (`L`). While the electrical network is connected to the mechanical part through an electromotive force as shown in Listing 4.3.

4.4.2 DC motor with gear train

The DC motor is an electromechanical component; consequently, its modeled with electrical and mechanical differential equations (see Listing 4.3). While the gear train component receives the torque from the

```
1  `include "disciplines.vams"
2  `include "constants.vams"
3  `timescale 1us / 1us
4  module motor(shaft, p, n);
5    // PARAMETERS ------------------------
6    // Motor constant (V-s/rad)
7    parameter real km = 4.5;
8    // Flux constant (N-m/A) = Kt
9    parameter real kf = 4.5;
10   // Inertia of the shaft (N-m-s2/rad)
11   parameter real js = 1.2;
12   // Drag rotating friction of the shaft (N-m-s/rad)
13   parameter real ds = 0.1;
14   // Motor winding resistance (Ohms)
15   parameter real r = 5.0;
16   // Motor winding inductance (H)
17   parameter real l = 0.02;
18   // Inertia of the wheel (N-m-s2/rad)
19   parameter real jw = 0.005;
20   // Drag rotating friction of the wheel (N-m-s/rad)
21   parameter real dw = 0.1;
22   // PORTS ---------------------------
23   output shaft;
24   input p, n;
25   // NODES ---------------------------
26   electrical p, n;
27   // Internal nodes.
28   electrical n1, n2;
29   rotational_omega shaft, rgnd;
30   // Reference nodes.
31   ground rgnd;
32   // BRANCHES ------------------------
33   branch (p, n1)  Vm;
34   branch (n1, n2) R1;
35   branch (n2, n)  L1;
36   branch (shaft, rgnd) bshaft;
37   // BEHAVIOR ------------------------
38   analog begin
39     // Electrical model of the motor winding.
40     V(Vm) <+ km * Omega(bshaft);
41     V(R1) <+ r * I(R1);
42     V(L1) <+ l * ddt(I(L1));
43     // Physical model of the shaft.
44     Tau(bshaft) <+ +kf * I(Vm);
45     Tau(bshaft) <+ -(ds + dw) * Omega(bshaft);
46     Tau(bshaft) <+ -(js + jw) * ddt(Omega(bshaft));
47   end
48 endmodule
```

Listing 4.3: DC motor model, fault-free.

connection with the DC motor, so it is modeled only with mechanical equations. The following constitutive relations model the dynamics of the motor connected to the gear train:

$$v_1 = K_E \cdot \omega_1 + R_A \cdot i_A + L_A \cdot \frac{d(i_A)}{dt}\,, \tag{4.10}$$

$$\tau_1 = K_T \cdot i - J_1 \cdot \frac{d(\omega_1)}{dt} - B_1 \cdot \omega_1\,, \tag{4.11}$$

$$\tau_2 = -N \cdot \tau_1 + J_2 \cdot \frac{d(\omega_2)}{dt} + B_2 \cdot \omega_2\,, \tag{4.12}$$

where v_1 is the input voltage source applied to the windings of the motor to control the velocity of the rotor; the variable *i* represents the current through the windings; the variable ω is the angular velocity of the shaft; the variable τ is the torque on the shaft. The value of each parameter used in this test case is listed in Table 4.7. K_E and K_T are motor coefficients used to dimension the size of the dc motor. The rotor and the shaft are assumed to be rigid, and the friction torque is proportional to the angular velocity of the shaft. The Equation (4.10) and the Equation (4.11) represent the electrical and mechanical rotational dynamics of the dc motor. While Equation (4.12) represents the dynamic of the gear train that receives the opposite torque (turns in the opposite directions of the motor) from the motor shaft reduced by the factor N representing

the reduction factor. This mechanical system is used in many applications to increase the torque on the gear train shaft, e.g., as the driver of the joints of an anthropomorphic manipulator [182]

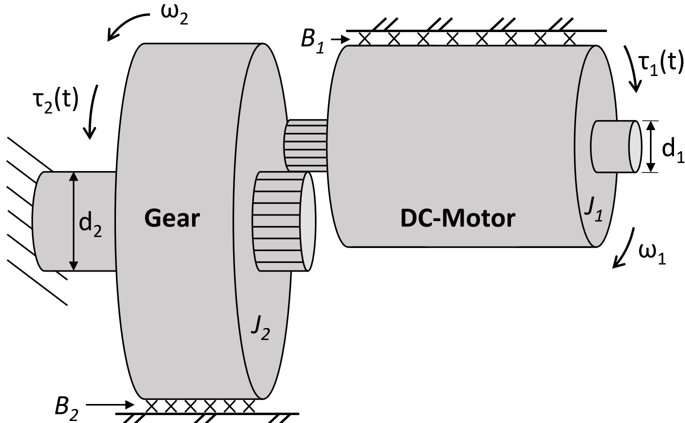


Fig. 4.20: DC motor with a gear train, the dc motor (on the right top of the figure) is connected with the gear train (center of the figure).

4.4.3 Multi-domain fault injection and simulation

The DC motor with gear train has been modeled with the Verilog-AMS language through two different modules as presented in Listing 4.4. The first module contains the DC motor equations (Equation (4.8) and Equation (4.9)), while the second module includes the gear train equation (Equation (4.12)). The two modules are interconnected by a `rotational_omega` port that allows the torque exchange between the two components. The simulation environment is set up on a CentOS machine built of an i7-9700 with 3.0 GHz frequency and 16 GB RAM memory. The Verilog-AMS code is simulated with the Questa-ADMS tool (that uses Eldo as an analog simulator and Questa as a digital simulator) of Siemens EDA.

Table 4.7: DC motor with gear train parameters.

Variable	Value	Unit measurement
K_E	0.1785	$V/1000rpm$
K_T	141.6 $\cdot K_E$	$in \cdot oz/Amp$
R_A	8.4	Ω
L_A	0.0084	H
J_1	0.0035	$in \cdot oz \cdot s^2/rad$
B_1	0.064	$in \cdot oz \cdot s/rad$
J_2	0.035	$in \cdot oz \cdot s^2/rad$
B_2	2.64	$in \cdot oz \cdot s/rad$
v_1	120	V
r_1	0.02	m
r_2	0.16	m
N	r_2/r_1	$ratio$

```
1  `include "disciplines.vams"
2  `include "constants.vams"
3  `timescale 1us / 1us
4
5  module motor(shaft, p, n, shaft2);
6      // PARAMETERS -----------------------
7      parameter real J1 = 0.0035;
8      parameter real B1 = 0.064;
9      parameter real Ke = 0.1785;
10     parameter real Kt = 141.6*Ke;
11     parameter real Ra = 8.4;
12     parameter real La = 0.0084;
13     parameter real J2 = 0.035;
14     parameter real B2 = 2.64;
15     real r1 = 2;
16     real r2 = 16;
17     real N  = 1;
18     // PORTS ---------------------------
19     output shaft;
20     input p, n, shaft2;
21     // NODES ---------------------------
22     electrical p, n;
23     // Internal nodes.
24     electrical n1, n2, fault;
25     rotational_omega shaft, shaft2, rgnd;
26     // Reference nodes.
27     ground rgnd;
28     // BRANCHES ------------------------
29     branch (p, n1)  vm;
30     branch (n1, n2) ra;
31     branch (n2, n)  la;
32     branch (shaft, rgnd) comp1;
33     // BEHAVIOR ------------------------
34     analog begin
35         N = r2 / r1;
36         // Electrical motor internal dynamic
37         V(vm)  <+ Ke * Omega(comp1);
38         V(ra)  <+ Ra * I(ra);
39         V(la)  <+ La * ddt(I(la));
40         // Motor dynamic
41         Tau(comp1) <+ + (Kt * I(vm));
42         Tau(comp1) <+ - B1 * Omega(comp1));
43         Tau(comp1) <+ - (J1 * ddt(Omega(comp1)));
44         Tau(comp1) <+ + (r1) * (Tau(shaft2));
45     end
46 endmodule
47
48 module reductor(tau_drive, tau_load);
49     // PARAMETERS -----------------------
50     parameter real J2 = 0.035;
51     parameter real B2 = 2.64;
52     real r1 = 2;
53     real r2 = 16;
54     real N  = 1;
55     // PORTS ---------------------------
56     output tau_load;
57     input tau_drive;
58     // NODES ---------------------------
59     rotational_omega tau_drive, tau_load, gnd_mec;
60     ground gnd_mec;
61     // BRANCHES ------------------------
62     branch (tau_load, gnd_mec) comp2;
63     // BEHAVIOR ------------------------
64     analog begin
65         N = r2 / r1;
66         // Gear train dynamic
67         Tau(comp2) <+ - (N * Tau(tau_drive));
68         Tau(comp2) <+ + (B2 * Omega(comp2));
69         Tau(comp2) <+ + (J2 * ddt(Omega(comp2)));
70     end
71 endmodule
```

Listing 4.4: Motor with gear train modeled in Verilog-AMS, fault-free.

The entire model has been simulated in different operating conditions: the faults from Table 4.6 have been tested on this system. Moreover, the system has been stimulated with different input curves, e.g., step or sine curve, in order to analyze the different faulty behaviors.

Some of those simulation results are shown in Figures 4.21, 4.22, 4.23, 4.24. The plots are related to the angular velocity of the DC motor with the gear train simulated for 10 seconds. They are related to fault-free simulation and four different fault models, one from the electrical domain and three from the mechanical domain. The electrical fault shown is the open, and it is injected into the electrical part of the model, which is in the motor. As for the mechanical ones, there is an external torque source on the motor with a value of $120N \cdot m$ (τ-source-m-120), a damper on the motor with a value of 1e03 (damper-m-1e03), and a torque source on the gear train with a value of $240N \cdot m$ (τ-source-r-240). Note that Figure 4.21 and Figure 4.23 are related to the simulation of the system stimulated with a voltage step of 120 V for 5 seconds. Furthermore, Figure 4.22 and Figure 4.24 are related to the simulation of the system stimulated with a sin curve of 120 V with frequency 0.1 Hz for 5 seconds.

The plots show how the fault-free response of the system (dotted blue line) reflects the behavior described by the system equations. Regarding the external torque source on the motor (red line), we can notice how the two components accelerate drastically. While the damper on the motor has the opposite effect with respect to the previous fault: the motor stops its rotatory motion. The external opposite torque source on the gear train (yellow line) forces the whole system to slow down the rotation. Finally, the open fault reduces the voltage feed for the motor, allowing the system to rotate less than the normal response.

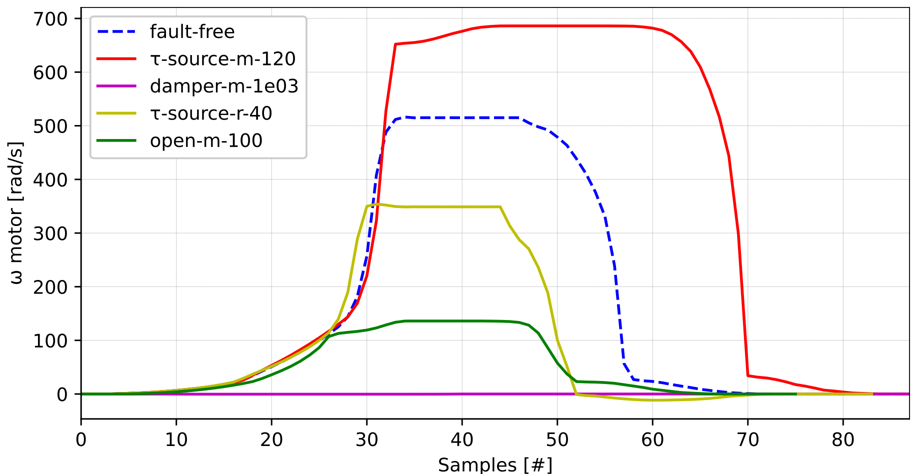


Fig. 4.21: Angular velocity of the motor, with a ramp as input.

We can notice how, feeding the system with different stimuli, faulty behaviors act in a different way. For example, with an inefficient supply voltage source to the motor, the effect of a damper on the motor cannot be visible because the motor would not move anyway. However, with a torque source on the gearbox, it is still possible to detect the faulty behavior of this fault (because everything should be stopped). Instead, with a high input voltage source to the motor, so the motor is working at high velocity, a torque source on the gearbox cannot be noticed, whereas the effects of a damper on the motor will be detected. For this reason, the presented simulation results have been obtained by feeding the system with stimuli that allow the detection of all the fault models injected. This type of analysis highlights how the quality of the testbench module affects the diagnostic coverage during the fault campaign simulation. These considerations take place in the testbench qualification analysis.

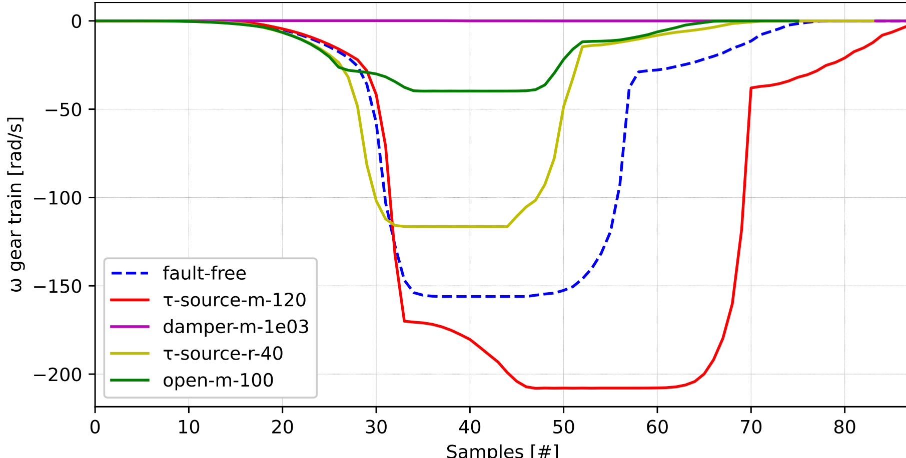


Fig. 4.22: Angular velocity of the gear train, with a ramp as input.

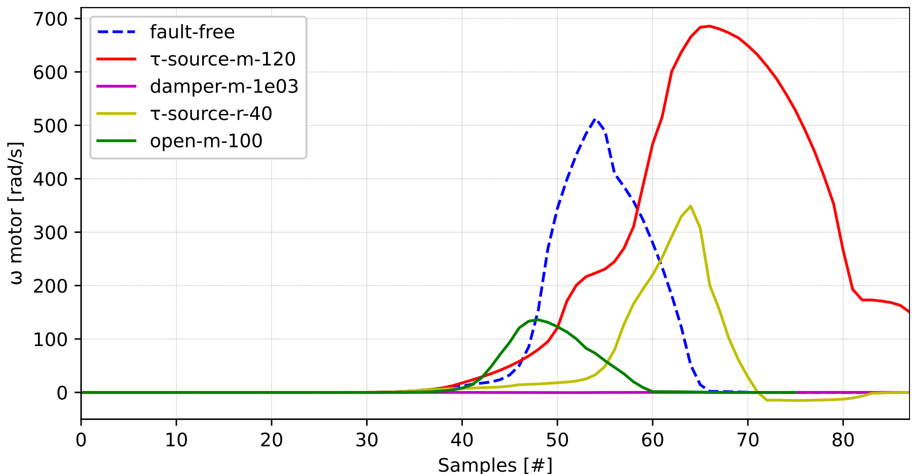


Fig. 4.23: Angular velocity of the motor, with a sinusoidal input.

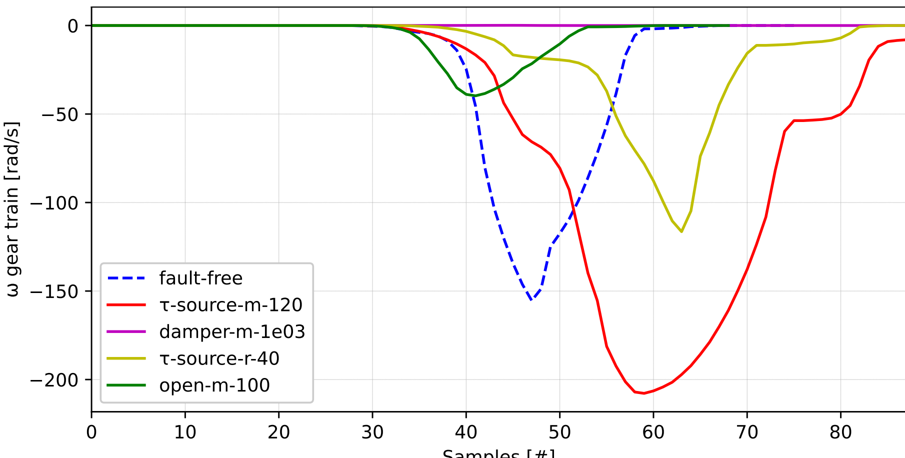


Fig. 4.24: Angular velocity of the gear train, with a sinusoidal input.

4.5 Automatic tool for fault injection and simulation

This section presents an engineering solution to automatize the fault injection into multi-domain models described with the Verilog-AMS language. Let us then take the DC Motor presented in the previous sections as an example extended with a wheel.

The system considered as running example is showed in Figure 4.25. The DC motor can be modeled with an equivalent circuit of the armature and the free-body diagram of the rotor. In details, the electrical part is composed by a voltage source that drive the current flow, a resistance connected in series with an inductance that model the internal characteristics of the internal motor coils. Then an electromotive force (emf) is used to model the conversion of the energy from the electrical to the mechanical domain. The mechanical rotational part model a shaft with own inertia connected to a wheel with a fixed inertia.

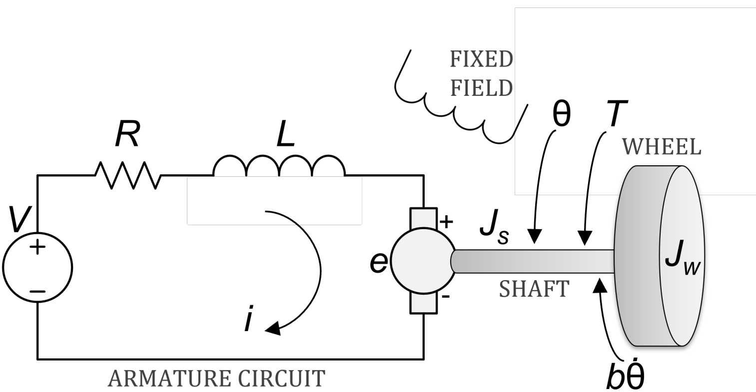


Fig. 4.25: DC motor plus wheel: the left part show the electrical circuit used to control the motor with a potential source, while the left part shows the shaft connected to a wheel. The EMF convert the energy from the electrical to the mechanical domain.

The following constitutive relations model the dynamic of the motor connected to a wheel:

$$v = K_M \cdot \omega + R_M \cdot i + L_M \cdot \frac{di}{dt}\,, \tag{4.13}$$

$$\tau = K_T \cdot i - (d_S \cdot \omega + d_W \cdot \omega) - (j_S \cdot \frac{d\omega}{dt} + j_W \cdot \frac{d\omega}{dt})\,, \tag{4.14}$$

where v is the input voltage source applied to the motor's windings to control the velocity of the rotor; variable i represents the current through the windings; variable ω is the angular velocity of the shaft; variable τ is the torque on the shaft. K_M and K_T are motor coefficients used to dimension the size of the motor: coefficient K_M is used to compare motors and to calculate temperature rise based on the dissipated power, while coefficient K_T is used to calculate the required current based on the required torque. The rotor and the shaft are assumed to be rigid, and the friction torque is proportional to the shaft angular velocity. According to the analogy, the inertia (j_S and j_W) and friction (d_S and d_W) of the shaft and the wheel in Equation (4.14) are equivalent to a capacitor (L_S and L_W) and resistance (R_S and R_W) respectively.

The electrical part provides power to the electric motor itself, feeding the power supply, while the mechanical section is composed of the shaft and a wheel, directly connected to the motor. In particular, it moves the wheel, which is connected directly to the shaft: these two mechanical parts cause resistance to the torque exerted by the motor, such as rotating friction and inertia. The system has been described through

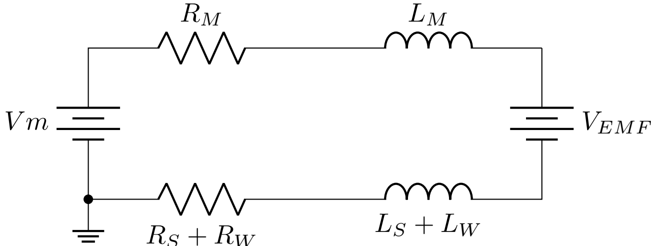


Fig. 4.26: Full electric description of the DC motor plus one wheel.

the HDL Verilog-AMS and simulated using the techniques and tools explained in the previous sections. The DC motor plus wheel model written in Verilog-AMS is shown in Listing 4.3.

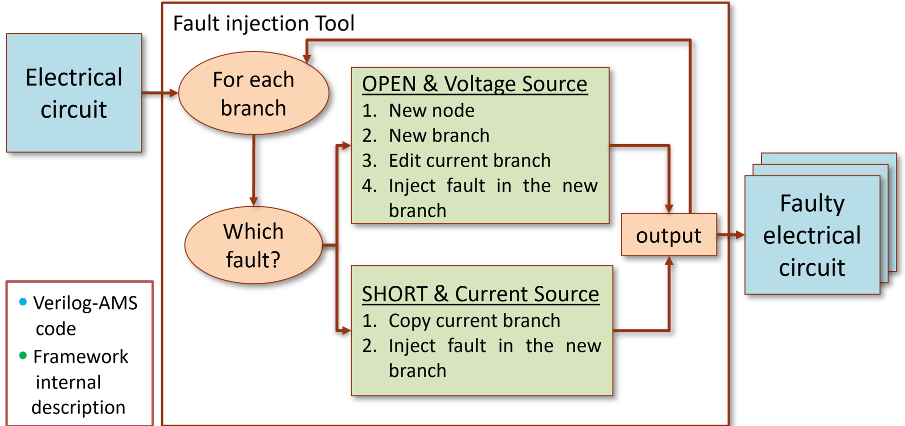


Fig. 4.27: Framework flow.

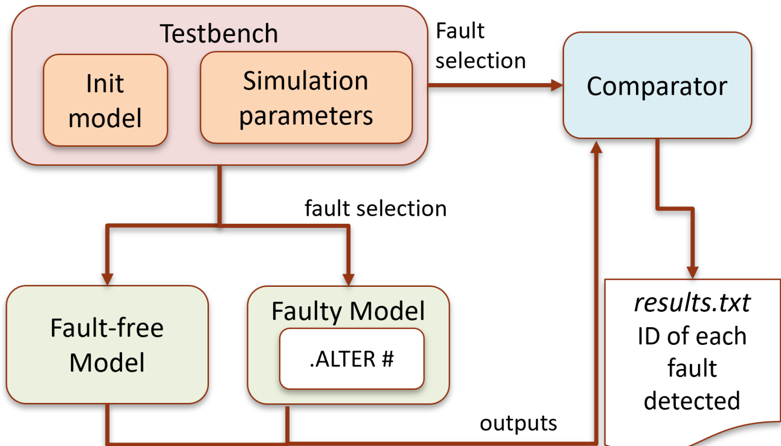


Fig. 4.28: Simulation flow.

As previously anticipated, the system was translated to electrical via the *force-voltage* analogy: the already electrical part remained unchanged, while the mechanical part was modeled as shown in Figure 4.26. This circuit was modeled in Verilog-AMS and is displayed in Listing 4.5. Note that the branches `Rsw` and

`Lsw` represent the combined values of the shaft and wheel resistors and capacitors, according to Equation (4.9) and Figure 4.26.

Once the electrical equivalent is obtained, we are ready to inject the electrical faults into the new circuit. As already explained, faults are injected one at a time, at only one point in the circuit, following the P2427 standard. An example of an open fault injected into the DC motor is shown in Listing 4.6. This makes it very easy to inject the faults proposed in Section 4.3 into Verilog-AMS descriptions following the electrical discipline. The injection of fault models into an electrical circuit follows precise and well-defined steps. For this reason, it is possible to automatize this operation once defined the faulty branch. So, once the branch has been identified, it will be sufficient to do some predefined actions to inject the desired fault. The execution of the described injector is shown by the Algorithm 1, which proceeds to inject in all branches the open, short, and sources faults.

Algorithm 1: Automatic fault injection algorithm.

Data: Nodes **N**, branches **B** composed by tuple of Nodes (n1,n2), the circuit itself **C**, and type of fault **T**
Result: List of faulty circuits

```
1  Create an empty list of faulty circuits FC;
2  foreach branch b in B do
3      Clone circuit C and modify it as CC;
4      if T is a short then
5          Add a new branch (n1,n2);
6          Add a uniquely name to the new branch short;
7          Add a short resistance in the new branch short;
8      else if T is a open then
9          Add a uniquely name node n3;
10         Modify the nodes of the branch b to (n1,n3);
11         Add a new branch (n3,n2) and a uniquely name to it open;
12         Add an open resistance in the new branch open;
13     else if T is a current source then
14         Add a new branch (n1,n2);
15         Add a uniquely name to the new branch current src;
16         Add a current contribution in the new branch current src;
17     else if T is a voltage source then
18         Add a uniquely name node n3;
19         Modify the nodes of the branch b to (n1,n3);
20         Add a new branch (n3,n2) and a uniquely name to it voltage src;
21         Add a voltage contribution in the new branch voltage src;
22     Add faulty circuit fc to the list FC;
```

Consequently, this is the behavior of the injector in relation to an electric circuit, equivalent to a mechanical system according to the analogy (see Section 4.2.2). However, it is not easy to alter a Verilog-AMS code, producing one that is functionally broken but syntactically correct. An intermediate circuit description is required to perform these operations. In detail, the proposed solution consists in passing to the software tool a Verilog-AMS model that realizes the electrical equivalent of the mechanical system to be failed. The tool then proceeds to transform the code into an intermediate representation that allows for the easy addition of new nodes, branches, and differential equations. After this step, according to the fault to be injected at the moment, the appropriate procedure will be executed.

```
1  `include "disciplines.vams"
2  `include "constants.vams"
3  `timescale 1us / 1us
4  module motor_el(shaft, p, n);
5    // PARAMETERS -----------------------
6    // parameters equivalent to the first model
7    // PORTS ---------------------------
8    output shaft;
9    input p, n;
10   // NODES ---------------------------
11   electrical p, n;
12   electrical n1, n2, n3, n4, n5;
13   rotational_omega shaft, wheel;
14   ground n;
15   // BRANCHES ------------------------
16   branch (p, n1)  Vm;
17   branch (n1, n2) Rm;
18   branch (n2, n3) Lm;
19   branch (n3, n4) Vback;
20   branch (n4, n5) Lsw;
21   branch (n5, n) Rsw;
22   // BEHAVIOR ------------------------
23   analog begin
24     V(Vm) <+ km * I(Vm);
25     V(Rm) <+ r * I(Rm);
26     V(Lm) <+ l * ddt(I(Lm));
27     V(Vback) <+ - kf * I(Vback);
28     V(Lsw)   <+ (ls - lw) * ddt(I(Lsw));
29     V(Rsw)   <+ (rs - rw) * I(Rsw);
30   end
31 endmodule
```

Listing 4.5: Full electrical DC motor model plus wheel, fault-free.

```
1  `include "disciplines.vams"
2  `include "constants.vams"
3  `timescale 1us / 1us
4
5  module motor_el(shaft, p, n);
6    // PARAMETERS -----------------------
7    // parameters equivalent to the first model
8    // PORTS ---------------------------
9    output shaft;
10   input p, n;
11   // NODES ---------------------------
12   electrical p, n;
13   electrical n1, n2, n3, n4, n5, x;
14   rotational_omega shaft, wheel;
15   ground n;
16   // BRANCHES ------------------------
17   branch (p, n1)  Vm;
18   branch (n1, x)  Rm;
19   branch (x, n2)  fault;
20   branch (n2, n3) Lm;
21   branch (n3, n4) Vback;
22   branch (n4, n5) Lsw;
23   branch (n5, n)  Rsw;
24   // BEHAVIOR ------------------------
25   analog begin
26     V(Vm) <+ km * I(Vm);
27     V(Rm) <+ r * I(Rm);
28     V(Lm) <+ l * ddt(I(Lm));
29     V(Vback) <+ - kf * I(Vback);
30     V(Lsw)   <+ (ls - lw) * ddt(I(Lsw));
31     V(Rsw)   <+ (rs - rw) * I(Rsw);
32     V(fault) <+ I(fault) * 1e09;   // open fault
33   end
34 endmodule
```

Listing 4.6: Full electrical DC motor model plus wheel, open failed.

4.5.1 Fault injection tool implemented inside HIFSuite framework

The best environment to develop this tool is HIFSuite (see Section 4.1.7). In fact, as Figure 4.29 shows, it already has the functionality to transform the HDL code in an intermediate description/language common to all tools in HIFSuite. The internal representation is structured hierarchically within an XML schema,

so the structure is tree-like. The HIFSuite framework and the injector proposed in this thesis are written in C++ language, allowing to obtain great efficiency in terms of allocated memory needed to manipulate Verilog-AMS descriptions and in terms of speed-up. This, in our case, is perfect since we are working on systems modeled in Verilog-AMS. This description works well for the systematic injection of the procedure described in the Algorithm 1. The injector will then be an additional tool in addition to HIFSuite: having received a Verilog-AMS file and transformed it into the intermediate description, the tool will inject all faults in each branch. All faulty modules are then generated and converted back to Verilog-AMS. By doing so, all behaviors of any injected faults can be studied and compared to the original. Furthermore, the same fault injection tool in the electrical domain can be easily extended to inject mechanical faults into mechanical models (not transformed into the electrical equivalent). This is possible because of the support that the Verilog-AMS language provides for many-domain models. The choice of the Verilog-AMS language is not accidental: besides allowing us to model systems described in different physical domains (disciplines in Verilog-AMS), it lets us manipulate the language in a systematic way, being a well-structured and complete language. Instead, modeling the same physical systems with other modeling tools, such as Simulink/Simscape and Modelica-based tools, a systematic and efficient fault injection would be more complicated.

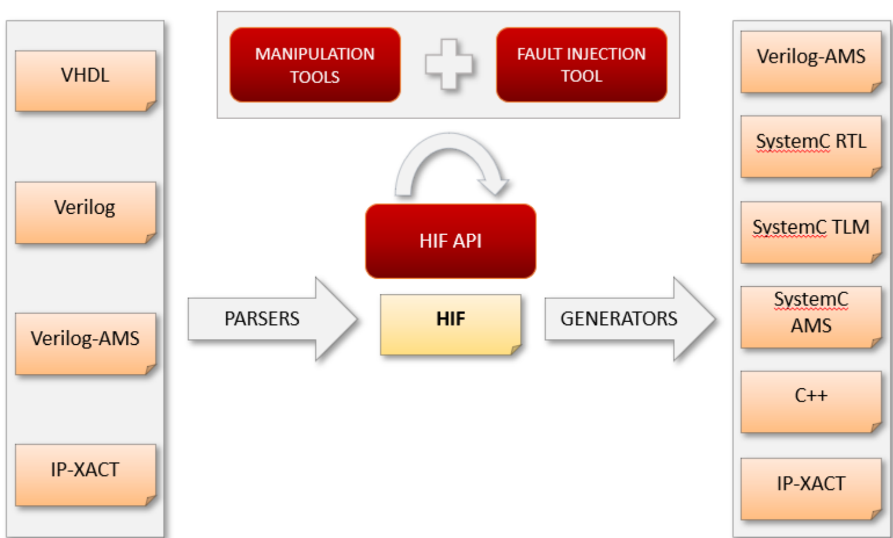


Fig. 4.29: HIFSuite architecture including the new fault injection tool.

The fault injection flow is also shown in Figure 4.27: the tool is able to take any circuit described in Verilog-AMS and translate it into XML as intermediate modeling for simpler fault injection. Then, depending on which fault we are injecting, there is one procedure for faults injected in series to the branch (i.e. open and voltage source) and one for faults injected in parallel (i.e. short and current source). Finally, all faulty circuits, according to the electrical fault models, are returned by the tool in their original HDL language. Once the automatic generation of the faulty models is performed, so all the faulty circuits are obtained. Regarding simulation, the process is handled by a testbench module, which instantiates the fault-free model (see Figure 4.28) and the faulty models through the `alter` command as shown in Listing 4.7. The comparator receives all the simulation output from the fault-free and faulty models; then, it provides as

Table 4.8: The average simulation time and the number of injected faults are reported for different systems.

Faults Data		RLC	Tuned-RLC	Double Pendulum	DC motor (mixed)	DC motor (full elec.)
Fault-free simulation time		570 ms	300 ms	190 ms	230 ms	240 ms
OPEN	No. injected faults	4	6	6	3	6
	Avg. simulation time	587 ms	241 ms	208 ms	250 ms	250 ms
SHORT	No. injected faults	12	30	-	12	30
	Avg. simulation time	610 ms	309 ms	-	240 ms	240 ms
Voltage Source	No. injected faults	4	6	6	3	6
	Avg. simulation time	4.8 s	1.28 s	1.5 s	450 ms	530 ms
Current Source	No. injected faults	12	30	30	12	30
	Avg. simulation time	1.5 s	2.1 s	1.4 s	380 ms	550 ms

output the fault pattern detected and at what point in the time. In this way, the simulation is run only once for all comparisons, writing all results to the dedicated file.

Appendix B present a series of Listings that shows the output of the fault injection tool, taking as input the codes of the fault-free models reported in the previous sections: the RLC circuit (Listing 4.1) and the Double-RLC circuit (Listing 4.2). Open, short, and voltage source faults are listed for the RLC circuit and the Double-RLC circuit, while only open and voltage source faults have been reported for the transformer. The current source has been omitted due to its similarity to the voltage source.

This section ends with a consideration: fault injection in Simulink/Simscape and Modelica may become non-trivial with certain classes of faults. In fact, adding a new physical factor to a differential equation in Verilog-AMS only requires adding a new contribution to an existing equation or adding a new equation in the physical network. As an example, even adding a Johnson–Nyquist noise to an electrical resistor in Verilog-AMS only requires adding new parameters that contribute to the equation. Instead, adding this noise to a predefined resistor component in both Modelica and Simulink is difficult unless the resistance is redefined as a new component block, which must be manually added to the library of available components. Another difficulty when using Modelica and Simulink is performing heterogeneous simulations, including the cyber part of a system, as it is not easy to combine a physical and mechanical system with a purely digital simulation that requires a cycle-accurate simulation [183, 184, 185]. This problem Verilog-AMS is automatically solved in Verilog-AMS because it natively supports digital, cyber, and even transistor-level descriptions [112].

Table 4.8 shows the injection and simulation results derived from different models introduced and illustrated in this chapter. Both the already presented versions of the motor are reported: the original electro-mechanical description and the analogous circuit. The table also shows the results of the same operations on other test cases, such as the RLC circuit, the Tuned-RLC (i.e., the electrical equivalent of the Mass-Spring-Damper) [170] and from the electrical equivalent of a double pendulum [186]. Specifically, the results refer to injection regarding the number of different faults (by position) injected into the circuit for each fault model. The results in the table also cover simulation: the time values reported are of both faulty and fault-free circuits. These times refer to a 30-second simulated behavior, and the circuit always receives the same input signal. Simulating faulty systems usually takes longer because faults add more overhead to system behavior.

```
1 .model motor macro lang=veriloga
2 * Instantiate Verilog-AMS motor
3 ymotor motor shaft p n
4 * Instantiate load resistor
5 rload n 0 1e09
6 * Initialize transient simulation
7 .tran 1u 1sec
8 * Define circuit alteration
9 .alter
10 ymotor motor_1 shaft p n
11 * List of all the faulty circuits
12 .end
```

Listing 4.7: Sketch of SPICE code to control the simulations.

4.5.2 Testbench qualification

The behavioral fault models presented in the previous sections are defined to be applied in analog descriptions that allow the representation of faulty continuous behaviors. This situation could present many variabilities due to the model itself and due to the stimulus provided to the model by the testbench. These stimuli can be applied at different points of the model, depending on the physical system under analysis. For example, a direct current motor can be stimulated by applying different source waveforms to the electrical part enabling a different response of the motor dynamic. This implies that different input waveforms can stimulate different faults injected into the model by changing the diagnostic coverage metrics calculated to asses the functional safety of complex systems. These variabilities due to the input waveforms are described in the literature for the analog domain as *testbench qualification* [187]. A general testbench structure is shown in Figure 4.30. The same concepts can be applied to systems described at the behavioral level, as proposed in this thesis. Consequently, an in-depth analysis should be performed to retrieve the range of values in which the input waveforms need to be positioned to stimulate the system correctly, e.g., by applying a specific waveform frequency range as input. By combining ad-hoc testbenches with systematical fault injection campaigns in multi-domain systems, correct diagnostic coverage metrics can be calculated to guarantee the functional safety of the overall system.

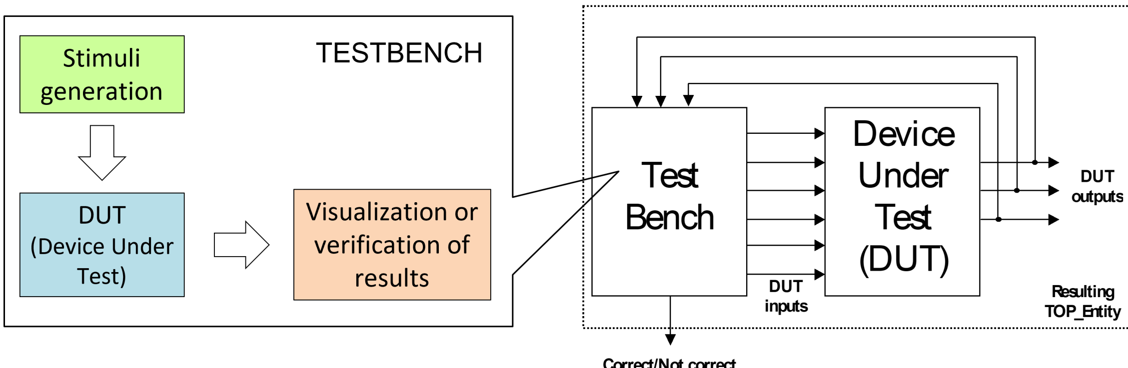


Fig. 4.30: Overview of the modules that compose a general testbench.

4.6 Inferring thermal fault models from the electrical domain

Machinery maintenance is the most important process in the context of smart factories, as the cycle of monitoring and fixing when necessary allows to avoid large losses of time and money. Additionally, this allows

for reduced downtime by keeping the machines in an efficient operational status, ensuring the *functional safety* of both the machines themselves and of the environment around them [188]. The level of functional safety of a system can be studied by first understanding when and how a fault behavior occurs.

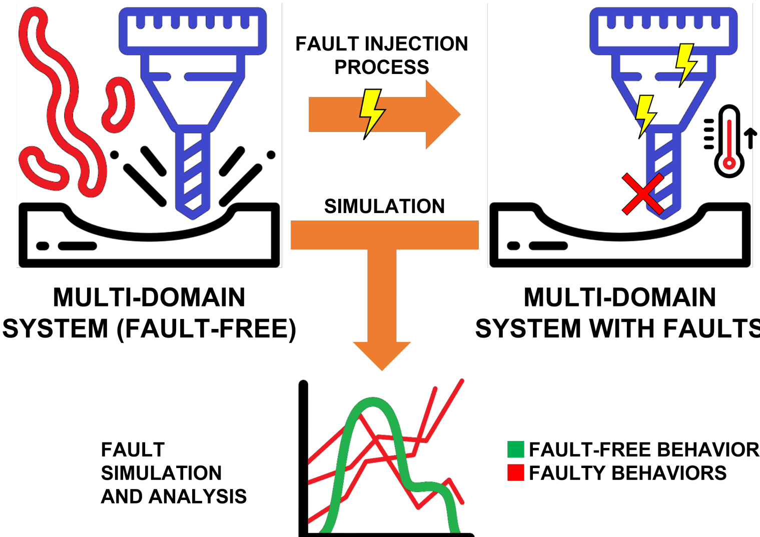


Fig. 4.31: Fault injection process in a multi-domain system, followed by the simulation of the fault scenarios.

As previously described, a well-known technique for studying the effects of faults on a system is *fault injection* and, afterward, fault simulation: by injecting well-defined fault patterns within a simulation model, the faulty behavior can be compared to the nominal one in a simulation environment under different scenarios and production conditions of both the machine under analysis and its surrounding environment. In the context of industrial CPSs, such scenarios take into account multiple physical domains, such as electrical, mechanical, and thermal, which typically influence each other [189]. Fault models typically refer to a specific physical domain, which means they will be injected into the functional model of a single domain (e.g., electrical or mechanical) by using a dedicated technique. Once the fault is injected, the relationships that already exist among the domains composing the system propagate changes even in domains other than the faulted one.

In the electrical domain, fault injection is state-of-the-practice for both analog and digital circuits, as reported in the ISO 26262 standard regarding functional safety for road vehicles [9, 8]. On the other hand, in the mechanical and thermal domains, fault classification and fault injection techniques are still at an earlier state. Mechanical models are more difficult to handle in their faulty versions: classification of these components is not always straightforward, and the fault taxonomies in the literature differ according to the geometry and material of the component.

While in the thermal domain, faults in the form of abnormal temperatures are often the consequence of some other disturbance, for example, electrical or mechanical variations. Extending fault taxonomies and fault injection techniques to these domains is a way to deepen the study of functional safety in more complex, multi-domain systems, and it is presented in this article. In order to accomplish this, the *physical analogies* existing between the electrical domain and the other domains can be exploited to bridge the differences between the physical domains involved, to bring to a common representation heterogeneous domains, and to ease the modeling of inter-dependencies. Furthermore, by describing a generic system

through an electrical circuit, the deep knowledge of this domain can be exploited for the *creation of new fault taxonomies and the simulation of faulty behaviors* belonging to different domains.

The main contributions of this work are:

- It describes the thermal model of a multi-domain system as an electrical circuit extended with faults by exploiting the physical analogies between thermal and electrical domains;
- It derives faulty thermal behaviors induced by thermal faults;
- It analyzes how a fault in a physical domain affects the other domains composing the simulated system;

Table 4.9: Physical analogies relationships between mechanical, thermal, and electrical domains.

Type	Rotational Mechanics	Thermal	Electrical
Effort	Torque	Temperature	Voltage
Flow	Angular Velocity	Heat Flow Rate	Current
Elements	Damping	Thermal Resistance	Resistance
	Moment of Inertia	-	Inductance
	Rotational Compliance	Thermal Capacitance	Capacitance

4.6.1 Physical analogies

The use of electrical circuits equivalent to other systems has become increasingly common over the past few decades [190]. The electrical domain is the best known compared to all others in many aspects: as a consequence, describing a non-electric system, e.g., mechanical or thermal, as an electrical circuit helps its modeling, understanding, and, in many cases, even simulation [191]. This process relies on the definition of physical analogies built from the mathematical equivalence of the laws belonging to the individual domains by mapping physical quantities and differential equations from one domain to another, with the purpose of mimicking the behavior through an equivalent electrical circuit.

Regarding the *mechanical domain*, there are two versions of the analogy with the electrical domain: force-voltage and force-current. The former is called the impedance analogy, and it assigns torque to voltage and angular velocity to current. The force-current analogy is also known as the mobility analogy, where torque is matched to current and angular velocity to voltage. The main difference is that force-voltage is known to be the most intuitive since the electrical effort and flux variables are equivalent to the mechanical effort and flux variables, respectively, and for this reason, this is the analogy used in this work. In contrast, force-current is more conservative about the circuit structure.

Similar to the mechanical domain, it is possible to describe a *thermal system* as an electrical circuit, e.g., for representing the thermal model of a conical rotor motor [192] or of lithium-ion cells [193]. The analogy maps temperature onto the potential difference between two nodes and heat flow rate to electric current [194]. It is easy to imagine how other electrical quantities are mapped equally in the mechanical and thermal domains: e.g., the electrical resistance is equivalent to the corresponding resistances in the other two domains. The relationships among physical magnitudes belonging to the electrical, thermal, and mechanical domains are shown in Table 4.9. Once modeled as an electrical circuit, a complex system can be quickly simulated and tested using modern simulators. It is important to note that simulating systems as equivalent electrical circuits is simpler at construction time and less expensive in terms of simulation resources than preserving the domain (or domains) of origin [191].

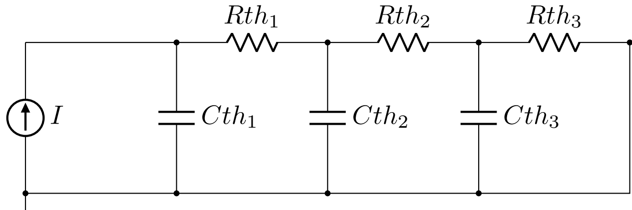


Fig. 4.32: Example of a Cauer network.

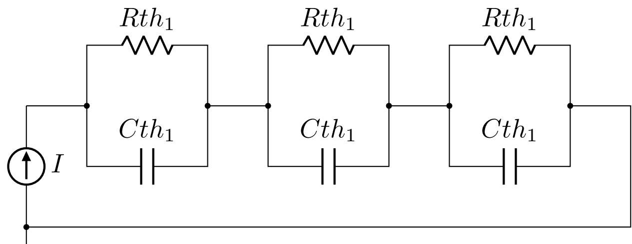


Fig. 4.33: Example of a Foster network.

4.6.2 Realizing thermal models through analogies

Temperature monitoring is critical for any class of industrial device, as it can change the inner parameters of the system, such as friction or electrical resistance. Moreover, temperature tracking is important for determining the operating conditions of the system and for the safety of the surrounding environment. For this reason, realizing the thermal model of a system under test is crucial.

Thermal analogous model

Several approaches exist for building thermal models by exploiting the analogies introduced in the previous section. The main thermal characteristics that determine temperature change are resistance and thermal capacitance. These two parameters can be intuitively considered equivalent to electrical resistance and capacitance. Moreover, according to the analogy, the temperature is equivalent to voltage while heat flow to electric current. Therefore, it is simple to consider an RC electrical network as an equivalent thermal model, and many thermal simulators rely on this assumption [194, 195, 196, 197, 198, 199]. These equivalent electrical circuits are known as Foster and Cauer networks [197]. The two networks, depicted in Figures 4.32 and 4.33, are equivalent in implementation (they can be easily transformed into each other), but they have different characteristics [200]. Creating these networks allows describing the thermal behavior of various classes of systems since the temperature is described as an extra-functional property and modeled in the same way through couples of resistors and capacitors.

In the *Cauer network*, all capacitors are grounded to force heat to flow only in one direction, and each node in the circuit represents a different temperature value. Specifically, voltage on the capacitor nodes represents the temperature at a specific point in the system, while ground represents the ambient temperature. The potential difference between a node and ground represents temperature difference between the object and the environment, and resistance consists of the amount of heat that can flow through an object or a solid surface. For example, in a model related to transistors, each resistance represents the constructive and material differences of each layer of the component. However, the Cauer network is difficult to describe mathematically, so using it to extract parameters such as capacitance and resistance is complex.

In the *Foster network*, only the last capacitor is grounded; all the others are connected in parallel with the resistors. This network has no physical meaning from a thermal point of view, as the nodes do not respect

the structure of the system (in the thermal domain, capacitance cannot take negative values). Nonetheless, the Foster network is useful for the extraction of resistance and heat capacity parameters and for making some analytical calculations.

For the reasons listed, the Cauer network representation was chosen in this paper as the equivalent thermal system to study the thermal behavior of the case studies. Furthermore, the accuracy of the thermal networks described in some articles as a Lumped-Parameter Thermal Network (LPTN) is proved to be comparable to a Finite Element Model (FEM) in the prediction of the effective temperature [201].

Modeling a real case of study: direct current motor

The case study adopted in this work is a DC-Motor (Figure 4.34), a well-known machine used in the industrial and automotive fields as it allows to control the speed of industrial applications precisely. A DC-Motor, also known as a direct current motor, is an electric machine that converts electrical energy into mechanical energy through the generation of a magnetic field powered by a direct current. Upon activation, the stator generates a magnetic field that interacts with the magnets on the rotor, causing it to rotate through attraction and repulsion. The commutator linked to brushes connected to the power supply delivers current to the motor's windings in order to maintain continuous rotation. One of the reasons why the DC-Motor is essential for industrial applications is its ability to precision control their speed. Now let us analyze the system: it includes an electrical actuating part, which is responsible for powering the system, and a mechanical part, which develops the torque generated by the motor. The electrical part can be modeled through a circuit consisting of a voltage source conducting the current through the system, a resistor, and inductance connected in series. These two parameters represent the intrinsic resistance and the inductance of the armature. The electromotive force produced by the internal coils of the motor is used to show the conversion between electrical and mechanical energy, expressed as torque. In this case, the motor is connected to its own shaft ending with a fixed gear for connecting a transmission system. The equations show the electrical and mechanical behavior of the DC-Motor:

$$v = K_M \cdot \omega + R_M \cdot i + L_M \cdot \frac{di}{dt} \tag{4.15}$$

$$\tau = K_T \cdot i - d_S \cdot \omega + -j_S \cdot \frac{d\omega}{dt} \tag{4.16}$$

where v is the input voltage of the motor. If this value increases, the angular velocity ω of the motor shaft will rise. K_M is the back electro-motive force (emf) coefficient, which represents the relationship between the back emf and motor speed. R_M is is the value of the armature resistance to electrical current, while L_M the value of its inductance. The electrical current around the motor circuit is expressed with i. Regarding the mechanical equation, τ represents the torque produced by the motor on its shaft. K_T is the torque coefficient, which determines the slope of the torque-current curve. This value is dependent on many characteristics of the motor structure, including magnetic strength, the number of wire turns, and the armature length. The parameters d_S and j_S stand for the friction and the inertia caused by the rotation of the shaft.

In the thermal part of the model, a Cauer network is used as equivalent thermal model of the motor (Figure 4.35). Specifically, the network consists of two resistor-capacitor pairs, modeling (1) the difference between the internal temperature of the motor (rotor) and the shaft and (2) the temperature difference between the shaft and the surrounding environment. These two pairs of resistor-capacitor represent a discretization into several regions of our system, belonging to different functional parts [202]. Power dissipation (P_{loss}) is the key value for calculating the input current in the Cauer network, and it is calculated as the current

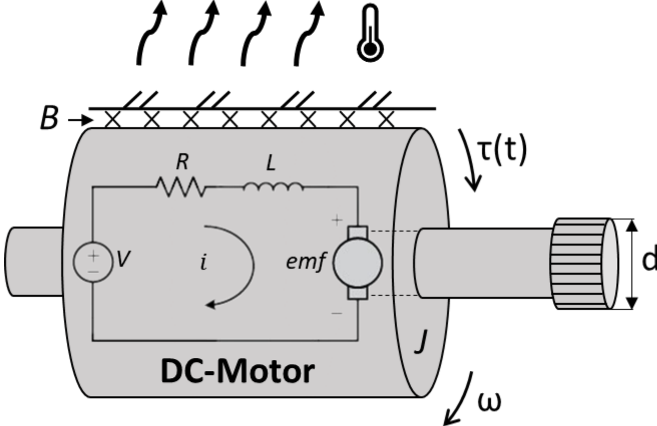


Fig. 4.34: A DC Motor, an example of a multi-domain system (electrical, mechanical, thermal).

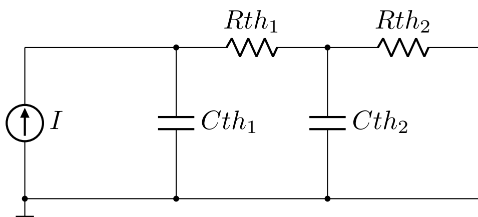


Fig. 4.35: Cauer network as the thermal model of the DC-Motor.

flowing in the motor times the armature resistance, as shown in the Equation (4.17):

$$P_{loss} = i^2 \cdot R_M \tag{4.17}$$

$$R_M = R_{M(0)} \cdot \left(1 + t_{winding} \cdot (T - T_{(0)})\right) \tag{4.18}$$

$$K_T = K_{T(0)} \cdot \left(1 + t_{magnet} \cdot (T - T_{(0)})\right) \tag{4.19}$$

The heat produced by this power turns into current following the analogy. Equations (4.18) and (4.19) show how the armature resistance value and torque constant vary as the motor temperature T changes. In the equations, the variables specified with (0) represent the initial value associated with the variables themselves. The values $t_{winding}$ and t_{magnet} indicate the temperature coefficients of the motor windings, made of copper, and the permanent magnet, considered to be composed of ceramic material.

Listing 4.8 shows the Verilog-AMS code of the proposed system as a case of study. There are two modules: one implementing the motor itself, with its electrical and mechanical equations (lines 5–54), and the other realizing the Cauer network as a thermal module (lines 56–80). The branch `temp` represent the feedback of the thermal module on the parameters of the main system.

```
1  `include "disciplines.vams"
2  `include "constants.vams"
3  `timescale 1us / 1us
4
5  module motor(shaft, p, n, ploss, gnd);
6
7      // PARAMETERS ----------------------
8      // Motor parameters
9      parameter real J1 = 0.0035; // shaft's inertia
10     parameter real B1 = 0.064; // shaft's friction
11     // Electrical/mechanical relations
12     parameter real Ke = 0.1785; // Back-emf coefficient
```

```
13     parameter real Kt = 141.6*Ke; // Torque coefficient
14     parameter real Ra = 8.4; // Armature resistance
15     parameter real La = 0.0084; // Armature inductance
16     // Thermal model parameters
17     parameter real Ta = 22; // Ambient temperature
18     parameter real tco_wind = 0.004; // Winding coeff.
19     parameter real tco_magn = -0.002; // Magnet coeff.
20     // DYNAMIC COEFFICIENTS ------------
21     real dyn_Ra, dyn_Kt;
22     // PORTS ---------------------------
23     inout  ploss, gnd;
24     output shaft;
25     input p, n;
26     // NODES ---------------------------
27     electrical p, n;
28     // Internal nodes.
29     electrical n1, n2, ploss, gnd;
30     rotational_omega shaft, rgnd;
31     // Reference nodes.
32     ground rgnd, gnd;
33     // BRANCHES ------------------------
34     branch (p, n1)  vm;          // motor vm
35     branch (n1, n2) ra;          // motor resistance
36     branch (n2, n)  la;          // motor inductance
37     branch (shaft, rgnd) comp;  // motor
38     branch (ploss, gnd) temp;   // temperature
39     // BEHAVIOR ------------------------
40     analog begin
41         // Electrical motor internal dynamic
42         V(vm)  <+ Ke * Omega(comp1);
43         V(ra)  <+ dyn_Ra * I(ra);
44         V(la)  <+ La * ddt(I(la));
45         // Motor dynamic
46         Tau(comp) <+ (dyn_Kt * I(vm));
47         Tau(comp) <+ - (B1 * Omega(comp));
48         Tau(comp) <+ - (J1 * ddt(Omega(comp)));
49         // Ambient temperature + power loss
50         V(temp) <+ Ta + (pow(I(vm),2) * Ra) * I(vm);
51         // Temperature effects on motor parameters
52         dyn_Ra = Ra * (1 + tco_wind*(V(temp) - Ta));
53         dyn_Kt = Kt * (1 + tco_magn*(V(temp) - Ta));
54     end
55 endmodule
56
57 module thermal(ploss, gnd);
58
59     // PARAMETERS ----------------------
60     parameter real R_th1 = 2.2;
61     parameter real C_th1 = 4.1;
62     parameter real R_th2 = 6;
63     parameter real C_th2 = 10/r2;
64     // NODES ---------------------------
65     electrical ploss, t1, t2, gnd;
66     inout ploss, gnd;
67     // BRANCHES ------------------------
68     branch(ploss, t1) in;
69     branch(t1, t2)  R1;
70     branch(t1, gnd) C1;
71     branch(t2, gnd) R2;
72     branch(t2, gnd) C2;
73     // BEHAVIOR ------------------------
74     analog begin
75         I(R1) <+ V(R1) / R_th1;
76         I(C1) <+ ddt(V(C1)) * C_th1;
77         I(R2) <+ V(R2) / R_th2;
78         I(C2) <+ ddt(V(C2)) * C_th2;
79     end
```

```
80 endmodule
```

Listing 4.8: Verilog-AMS modules implementing the DC motor and the Cauer Network module.

4.6.3 Thermal fault classification

Although the analogies allow the creation of valid models for systems belonging to different domains, this is not guaranteed concerning faults added as additional differential equations. A fault belonging to one domain may not have a physically valid and functionally relevant meaning in another domain, even if the purely mathematical behavior is the same. A physical meaning of the obtained equivalent fault model has to be investigated anyway to validate the mapping between the two fault models belonging to different domains.

An extension of faults from the electrical to the mechanical domain has been proposed in [203]. This paper thus focuses on a possible mapping between faults of the electrical domain into the thermal domain.

Due to the ability to model thermal systems as electrical circuits described in the previous section, injecting electrical faults into the equivalent thermal circuit and studying their physical effects on the whole system seems very promising. Then, the focus of this paper is not only to study the direct thermal effects of a thermal fault but also to understand what consequences it may have in a multi-domain system, such as the DC-Motor presented in the previous section.

The mapping research between electrical and thermal fault models is first performed on isolated RC Cauer networks of various sizes. Fault injection has been executed on Cauer networks consisting of one to three resistor and capacitor pairs to understand whether the electrical fault actually could have a physical meaning as an equivalent thermal fault. Once the mappings between faults were determined, the analysis switched to a real model, as explained in Section 4.6.4, i.e., where the Cauer network is supplied by the power loss of a real system.

Let us now present the mappings obtained between the electrical fault models and the thermal domain. Consider the analog faults proposed by the IEEE P2427 draft standard [32]: the analog faults proposed by this standard must be injected at a single point in the circuit, one fault model at a time. In this way, studying the variation of system behavior based on that fault model is feasible.

External heat source

one of the electrical faults proposed in the standard is the *current source* injected in a branch. This fault represents an unintended variation of electric current flow in a branch of the circuit. The cause of this fault may be due to external interference coming from adjacent branches or an external electromagnetic field. Referring back to the analogy, the coupling between electric current and heat flow is crucial when analyzing this fault model. Thus, by injecting electrical current into a branch of a Cauer network, the amount of current flowing in the circuit increases, compared to the current flow generated by power dissipation. Hence, at the thermal level, the system is exposed to a bigger heat flow than normal, causing a rise in the temperature of the component corresponding to the faulty branch. In general, the temperature rise in a mechanical component causes thermal dilatation of the materials composing the object, inducing internal stresses when subject to constraints [204]. Moreover, this anomaly temperature variation is usually a consequence of friction, defined as a passive source of heat that causes an increment in the system temperature, and a mechanical dilatation of materials.

Open Fault

as a resistance value higher than expected in a branch of the circuit. If the additional resistance has a very high value, the fault model simulates a line break in the circuit. The open fault increases the resistance value on a branch: from a thermal point of view, it represents an increase in the thermal resistance value of the faulty component. This effect can be caused by a sudden change in the physical properties of the material due to non-optimal physical working conditions of the system.

Parametric

as the specific internal parameters change, the behavior of the system changes. In the case of a Cauer network, if the resistance value changes, the current flow decreases at that point of the circuit. Thus, the thermal resistance expressed by the circuit is altered, changing the physical properties of the material of the faulty component. One possible reason for this fault could be a production defect or a deterioration of the surface that no longer insulates heat in the way it was meant to. The same fault can also be applied to the capacitor, thus changing the heat capacity value of the faulty component's material.

The discussed fault patterns also have consequences on the components dissipating power by their usage. Those effects, which are external to the thermal domain, are analyzed in the next section by showing faulty simulation traces. The previously described thermal faults must be considered in a fault injection campaign for a multi-domain system because it reveals new fault scenarios and new fault modes for the system under test. Regarding the electrical domain, a real fault mode is the damage to an electrical circuit, changing its resistive and capacitive properties, due to temperature growth. It is very well known that overheating of electrical components deteriorates their performance [204]. The same scenario is also potentially dangerous mechanically, as anticipated earlier. The deformation of a mechanical component due to high temperature seriously changes its behavior. These new fault modes expand the possible set of failures that a mechanical system could encounter during its normal function due to different physical causes.

4.6.4 Experimental validation

In this section, the DC-Motor introduced in Section 4.6.2 is considered as a running example to show the results of the fault injection and simulation process.

The system has been faulted in its thermal component by applying current sources, open faults and varying the equivalent thermal parameters that characterize the motor. However, other fault models have been applied to the system, not belonging to the thermal domain. Specifically, electrical faults have been injected into the original electrical component of the motor, and mechanical faults into the respective components. This multi-domain fault analysis is performed to understand what kind of influence a fault belonging to a single domain has on all others. As anticipated in the previous section, one fault at a time has been injected into the system, thus simulating individual instances of motor fault situations. The fault models that have been applied for this case study, grouped for the different domains, are:

- **electrical** [205]: short, open, voltage source, current source, parametric;
- **mechanical** [203]: external damping effect, component disconnection, external force, limited movement, parametric;
- **thermal**: external heat source, open thermal fault, parametric.

The results of this multi-domain failure analysis are shown in the Figures 4.37 to 4.39: Figure 4.37 depicts the angular speed of the motor, Figure 4.38 presents the temperature of the motor while Figure 4.39

shows the temperature on the shaft. Two faults observation points are considered in this analysis: the angular velocity of the motor and the temperatures at two points on the motor, one inside the motor and one of the shaft. The colors of the lines are common to all three graphs and represent the same scenario. The blue dashed lines represent the fault-free behavior of the system, which is stimulated with waves of different intensities: 12, 24, 48, and 64 V.

Let us now analyze the presented fault scenarios, which are represented with continuous lines and colored differently based on the injected fault model.

Regarding thermal faults, two models are shown in the figure: an abnormal heat source on the motor and an open fault, which increases the thermal resistance value of the motor. The first fault, named *thermal-source* and colored light blue, simulates the presence of an unwanted heat source on the motor, which impacts the mechanical movement of the motor by slowing its rotational speed. This effect is caused by the increase in friction between the mechanical components: the temperature of the components rises, while the shaft speed decreases. The change in internal motor parameters due to the temperature increase is shown in the Figure 4.36. Specifically, Figure 4.36 shows how an increase in motor temperature corresponds to an increase in power dissipation (instant 1 and 2 in the figure) due to the change in the motor's internal parameters. These parameters are the electrical resistance of the armature (`Ra`, green color), and the mechanical torque constant (`Kt`, red color). In particular, temperature growth increases the value of `Ra` and decreases the value of `Kt`. At the moment the motor's power supply runs out, the motor slows down to stop its rotation, and so the power dissipated decreases (instant 3 in the figure). As a result, the `Ra` and

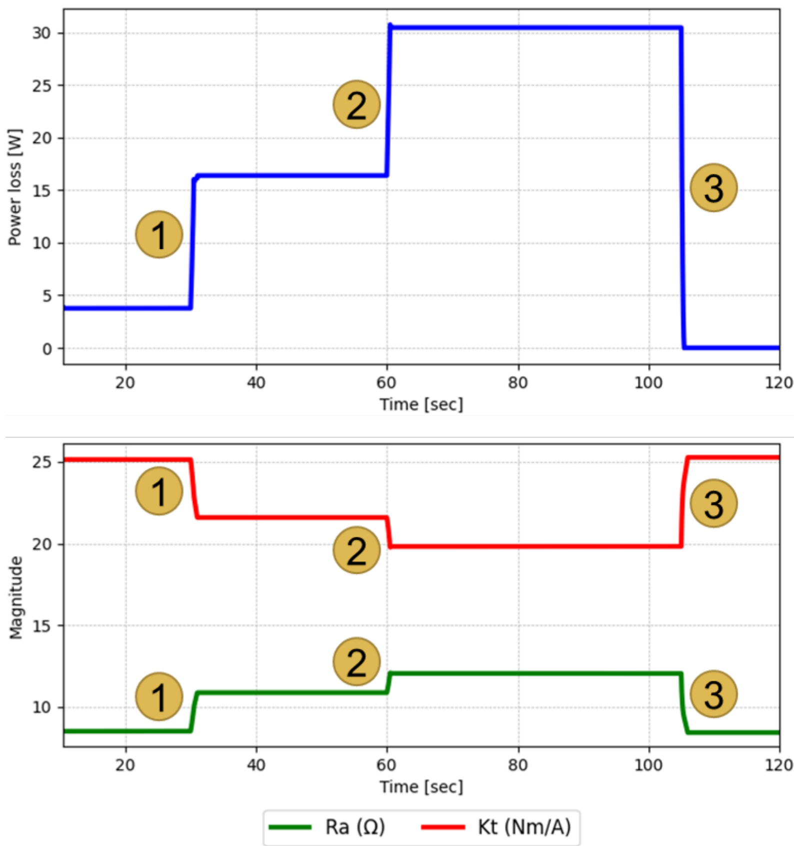


Fig. 4.36: Excerpt of the evolution of the DC-Motor power loss (upper graph) and internal mechanical parameters Ra (armature resistance) and Kt (torque constant) affected by an external heat source fault.

`Kt` parameters of the motor also return to normal. The injected heat source takes effect after 30 seconds of simulation for a value of 50 W.

The second thermal fault model, called *thermal-open*, with a value of 1000 K/s and represented by the yellow line, is a change in the thermal resistance value of the system, which is increased. This fault causes a drop in the temperature rise of the motor components since they are more insulated from the heat flow, while the angular velocity of the shaft remains unchanged with respect to the fault-free version. The reported effects of thermal faults on temperature affect mostly the mechanical domain, impacting the motion of the motor, and the electrical domain, increasing the power dissipated by it.

An open-type electrical fault, named *electrical-open*, has been injected with a resistance into the electrical part of the motor, simulating an increased resistance value of the motor's electrical circuit line. The resistor has been injected into the `ra` branch in the code shown in Listing 4.3, with the value of 1000Ω, and its response is highlighted by the red color. This fault causes a drastic drop in the power supply to the motor, which barely rotates, and the temperature of the motor and the shaft remains stable. Thus, an electrical fault model has direct effects on its own domain and consequential effects on the mechanical and thermal domains. This fault is injected by adding the following fault line in the Verilog-AMS code:

```
V(n1, n2) <+ I(n1, n2) * 1e03;
```

where `n1` and `n2` are the two nodes of the faulty branch. The open fault is injected in series to an existing branch, so one of the two nodes `n1` and `n2` has to be a new node, while the other is already existing.

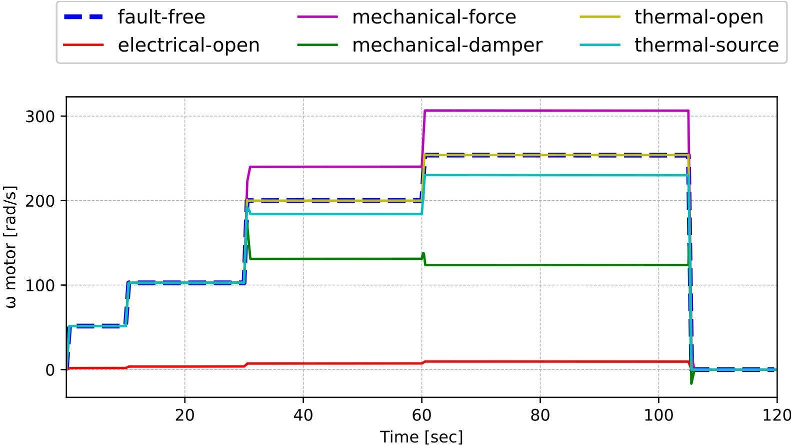


Fig. 4.37: Angular velocity of the DC-Motor in fault-free and faulty scenarios.

In addition, two mechanical faults were also injected and simulated: a *braking force* applied to the shaft and a force that contributes to the *shaft rotation*. In the first case, namely *mechanical-damper*, the fault causes the motor to slow down its rotational speed, causing a drastic increase in power dissipation and, thus, an increase in both motor and shaft temperatures. This faulty behavior is shown with a green line, and it has been injected after 30 seconds of the simulation and with a negative contribution of 50 Nm. In the second case, named *mechanical-force*, a positive contribution to the motor rotation has been injected. The motor rotates faster than it should: the higher this rotation is, the lower amount of power is dissipated, reducing temperature growth. The violet line in the plots shows how motor speed increases from the time of fault injection (50 Nm after 30 seconds), while shaft motor temperatures grow much more slowly. Again, faults belonging to one domain, mechanical in this case, change mechanical behavior as well as electrical and thermal ones. Simulations have been performed using the Verilog-AMS framework, instantiating the

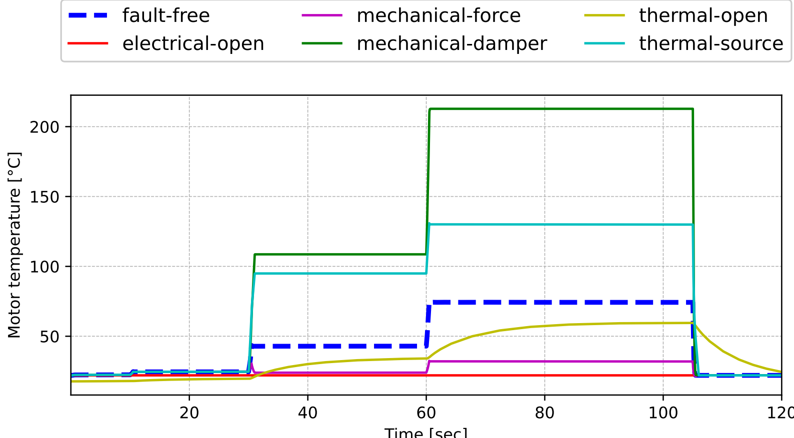


Fig. 4.38: Temperature of the DC-Motor's rotor in fault-free and faulty scenarios.

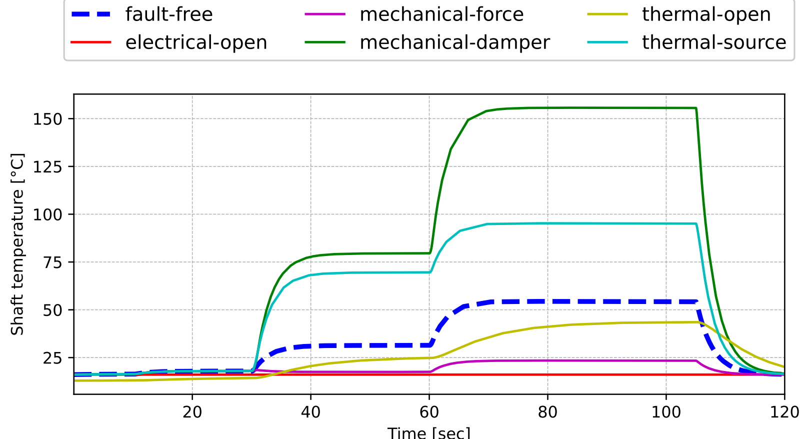


Fig. 4.39: Temperature of the DC-Motor's shaft in fault-free and faulty scenarios.

motor and Cauer network modules through a common testbench module. This top-level component powers the motor, providing the input voltage suitable for stimulating the applied fault correctly, with a given value, and at a specific time. The testbench has been built and refined during the analysis to stimulate faults not only in the motor but also in the Cauer network, for example, by injecting heat on the branches of the thermal network. This type of analysis is necessary to define an optimal testbench able to stimulate the system in the context of functional qualification of the verification testbench [187] even in a multi-domain system.

5

Model abstraction

Nowadays, with the advent of smart systems, the integration of analog and digital components into heterogeneous systems is crucial [206]. Generally, these heterogeneous systems composed of hardware descriptions and software applications are modeled as Virtual Platform (VP). A VP is a functional representation of a digital or analog system written entirely in software. This software is written to implement the modeled platform's specification rather than the detailed implementation of the circuits at HDL or RTL levels [207]. Complex embedded systems are usually modeled into the virtual platform using SystemC-AMS TLM or C++ languages. To model and integrate embedded systems into VPs, high-level descriptions of these components are required. These high-level descriptions of complex systems usually are not available for Integrated Circuit (IC) designers and generate it is an open research field. In this context, finding efficient methodologies to simulate heterogeneous VP at the functional level is fundamental. This chapter presents different methodologies that allow abstracting analog descriptions to behavioral descriptions. All the following abstraction methodologies are used for the deployment of digital twins to improve industrial systems' safety. The chapter is organized as follows:

- Section 5.1 proposes a methodology to abstract Piecewise Linear (PWL) descriptions written in Verilog-AMS to simulable to C++ code;
- Section 5.2 describes a methodology flow able to abstract analog descriptions (linear and non-linear). The final model is written in Verilog-A and communicates with a dedicated C++ library that contains the abstracted model. This flow is built for electrical descriptions, but it is also applicable for mechanical descriptions.

5.1 Abstraction of PWL models to C++

This section is focused on the functional level abstractions and simulations of Verilog-AMS analog descriptions (described as PWL). Verilog-AMS is a hardware description language (HDL) used principally to describe behavioral models of analog descriptions. It is possible to generate simulable C++ code from an analog description through the proposed methodology. Usually, analog systems are defined with non-linear behaviors, and ensuring simulation convergence is difficult. In this context, many techniques allow linearizing the non-linear systems in precise time points through mathematical techniques or simulation. This process allows to build PWL descriptions of non-linear systems and to perform efficient simulations on them. As shown in figure 5.1 the proposed methodology is applied to the PWL descriptions to build functional models. Starting from the parsing of all the information contained in a Verilog-AMS PWL model, all the information is stored in an Abstract Syntax Tree (AST). By visiting this tree and manipulating the equations, it is possible to symbolically solve all the systems of equations built from the original model.

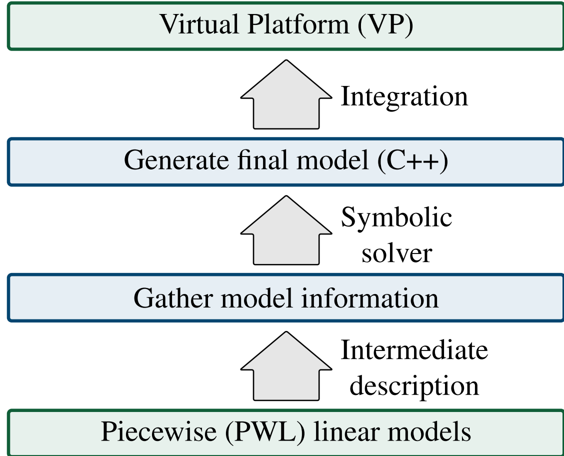


Fig. 5.1: Overview of the entire workflow.

Finally, the systems of equations solved are coded into the final C++ model and then integrated into a VP. The main contributions proposed in this section are:

- Definition of an automatic abstraction methodology from piecewise Verilog-AMS linear model to C++ model;
- Integration of the abstracted C++ models into VPs to speed-up the overall simulation time;
- Comparison of the simulation accuracy and speed-up between piecewise linear models described in Verilog-AMS and functional level models described in C++;

5.1.1 State of the art and definitions

This section explains how at the state of the art, it is possible to linearize non-linear models and generate a simplified PWL model. Finally, it presents a review of state-of-the-art related to abstraction methodologies of analog models.

Piecewise Linearization

Piecewise approximation is a method in which a function $g(x)$ is constructed to fits a objective function $f(x)$ that is non-linear. By adding extra variables (binary or continuous) and constraints, it is possible to reformulate the actual problem for the approximation of single-valued function in terms of linear segments sequence [208, 209]. A piecewise linear approximation will approximate a function $g(x)$ made up of a sequence of linear systems on the same intervals as the $f(x)$ function is defined. Usually, transistor-level simulators during the transient simulations compute the solution on the first time point and then linearize the system of equations to estimate the jump they will make to arrive at a subsequent time point. Using linearization, simulators can compute the direction towards which it has to move to satisfy all the equations at the next time point. For the first time, it usually takes the case with zero potential differences everywhere across the system (all inductors and capacitors discharged), which has a trivial solution of all currents equal zero. Then, the solver moves in small steps in the time domain. If the solver makes a step and then finds that at the new trial point it has computed, the circuit equations are not correctly solved (meaning there is a residual error due to the approximation in the linearization phase). Then, it will invalidate that trial point and try a new one, typically after reducing the time-step, making a smaller jump, and reducing the chance of making an error. It is fundamental to understand the behavior of the circuit at equilibrium points (DC

operating Points) [210]. A circuit can be linearized at any point, but typically solvers choose to perform it at specific equilibrium points.

State of the art of abstraction methodologies

Most of the work present in literature is related to the abstraction from transistor to behavioral level or vice-versa. Moreover, many of these works are related to the functional level abstraction of only linear circuits [211]. Conversely, in our work, we are dealing with PWL models of analog circuits derived from non-linear descriptions. This article is an extension of the methodology presented in [211] to enable the support of PWL descriptions. In [212] an automated abstraction technique has been presented. However, this technique was related to the abstraction of analog models described at the transistor level to the behavioral level. Another work that proposes an abstraction method from transistor-level models was discussed in [213]. In [214] a hierarchical abstraction methodology was explained. An optimum specification of analog circuits was translated from higher to lower-level abstractions to design the analog circuits.

5.1.2 Abstraction methodology

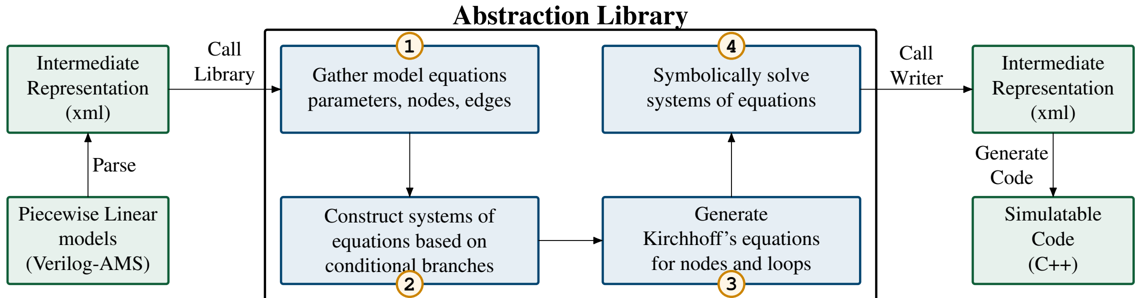


Fig. 5.2: Proposed methodology flow to abstract PWL descriptions.

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

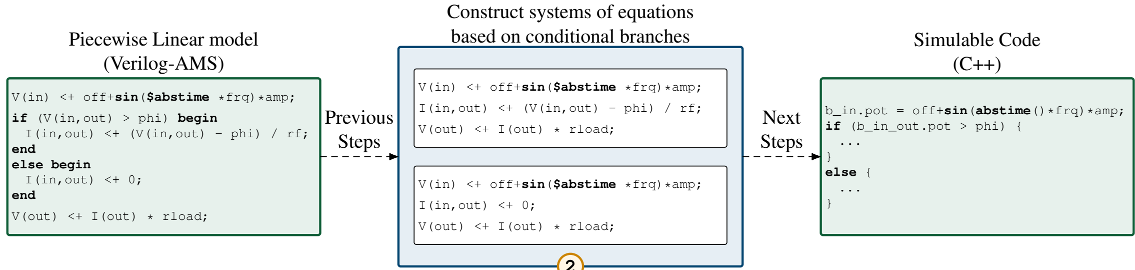


Fig. 5.3: Process flow used to abstract the model equations from Verilog-AMS to C++.

Software infrastructure

The software infrastructure required to abstract Verilog-AMS PWL models is described in Figure 5.2. The library that implements the methodology is built with the C++ language. The software infrastructure consists of the following steps built into different tools:

- The starting point is Verilog-AMS code that consists of both linear and non-linear equation models (see Listing 5.1). First, it linearizes the non-linear equations into piecewise linear models. Piecewise linear models can have both `if`/`else` and switch statements as shown in Listing 5.2.
- Then, the piecewise linear model and the linear models are parsed and transformed into an intermediate description. This intermediate description is an Abstract Syntax Tree (AST) written in XML that stores all the pieces of information of the original model by structuring them in a tree.
- When the information of the original model is stored inside the intermediate description, it is required to gather all the information to manipulate the model equations.
- Additional equations are computed through the Kirchhoff's current law (KCL) and Kirchhoff's voltage law (KVL) to generate the branch current and branch voltages equations.
- Then, the set of equations is simplified before solving them. Only *linearly independent equations* and their unknowns can be passed to the solver.
- In this step, we have a simplified set of equations and unknowns. Starting from it, the equations are symbolically solved through the GINAC C++ library [215]. Other libraries and tools can be used to solve the complete system of equations, like, MATLAB core or the SymPy library written in Python.
- The solved group of equations is re-written inside the intermediate description (XML file). Then, through a back-end, it is possible to generate the final abstracted C++ code. Moreover, the generated C++ code then is easily integrable into a VP.

Equations abstraction methodology

The entire abstraction methodology used to process and abstract the equations is schematized in the central part of Figure 5.2. Four main steps of the abstraction methodology can be identified:

- In the first step, the library takes in input all the equations, nodes, and branches of the original parsed model. The original model is written using Verilog-AMS language and contains an analog circuit's piecewise linear representation. An analog circuit can be seen as an electrical network composed of nodes, branches, and equations. Starting from the equations, all the `if – else` statements are identified and parsed. This is a fundamental step of the methodology because for each `if – else` statement identifies a different electrical network, and for each electrical network different equations are associated. This means that the original circuit expressed as an electrical network change internally what an `if – else` statement is present in the model (Figure 5.2 step 1).
- For each `if – else` statement the library build an electrical network for each `if – else` statement. The idea is similar to the concept of unrolling a loop inside the C++ language through the `#pragma unroll` statement. With the `#pragma unroll` statement, it is possible to speed up the code many times. In the same manner, with the proposed process of building a different electrical network for each `if – else` statement, it is possible to speed up the simulation. At the simulation time, the final C++ code containing the abstracted model enables one electrical network when the dynamic requires to change from a linear system to another (from one electrical network to another). While the commercial analog circuit simulator (based on SPICE) builds the system of the equation at run time, this means that the system of equations can change at run time when the execution of the model arrives at `if – else` statements (Figure 5.2 step 2). This second step is the core of the proposed methodology and is described through the equations of a half-wave rectifier in Figure 5.3.
- In the third step, all the different electrical networks (composed of different equations) are symbolically solved one at a time through the GINAC library. The equations passed to the solver are the original, KCL and KVL, and the unknowns. The KCL and KVL equations are computed starting from

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

Listing 5.1: Non-linear model of a half-wave rectifier written in Verilog-AMS.

the original equations for each node and branch of the electrical network. If the original model contains equations outside the `if − else` statement, these equations are duplicated inside each different electrical network; this is necessary when we solve the equations symbolically because the final model generated depends on all the components of the system (Figure 5.2 step 3).

- While, in the last step, the final C++ model is built starting from the solved equations. In the final code, a different system of equations for each electrical network is present. After the solving phase, if a component should be added to the electrical network, all the abstraction methodology should be re-run because the solved equations depend on the structure of the abstracted analog circuit. When the final C++ model is simulated, only one electrical network is active in a precise simulation time. This means that the final model is extremely fast because described in the C++ language and because all the possible systems of equations of the model are explicated before the execution of the model (Figure 5.2 step 4).

5.1.3 Methodology validation

In this Section, the methodology is validated through three PWL analog models. The proposed methodology is implemented into the HIFSuite [216] framework written with the C++ language.

Analog models analyzed

The methodology is validated through three analog PWL models:

- A half-wave rectifier taken from the book [181] that allows only one half-cycle of a voltage waveform to pass, blocking the other half-cycle;
- A memristor taken from the article [217] that model a non-linear component that links the electric charge and the magnetic flux natures;
- An ideal relay taken from the book [181].

Both of these models are simulated with a common testbench (see schematic in Figure 5.5). The testbench is modeled with a voltage source connected to the model and a load resistance to improve the convergence of the simulation.

5.1.4 Methodology validation

The entire methodology is validated with a model of a half-wave rectifier. The model is composed of a voltage source, a diode, and load resistance. The diode is the main component in this model, and its behavior is non-linear. A diode works in two regions named: forward biased and reversed biased. A PWL is used for the approximations of the diode characteristic curve in the form of linear segments. First, we have checked the accuracy between both non-linear and PWL models of the half-wave rectifier to check the correctness of the PWL model. For this step, both models were simulated on a SPICE-based simulator. After that, the

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

Listing 5.2: Verilog-AMS PWL model for a half-wave rectifier.

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

Listing 5.3: Verilog-AMS top level.

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

Listing 5.4: C++ analog branch structure.

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

Listing 5.5: C++ abstracted code of half-wave rectifier. Variable name that start with 'b_' are of type *analog_branch_t*.

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

Listing 5.6: C++ simulation manager.

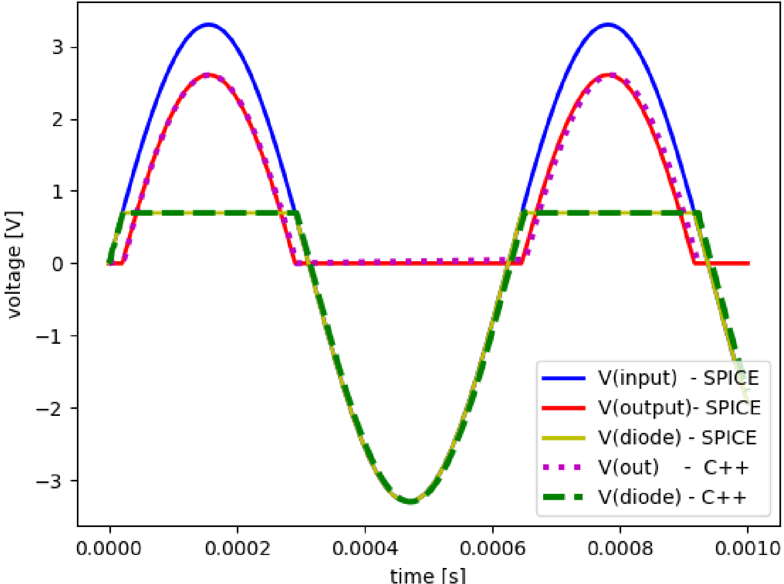


Fig. 5.4: Simulation results for the half-wave rectifier, the memristor, and the ideal relay.

abstraction methodology is applied to the half-wave rectifier model. The abstraction flow applied on the model equations is schematized in Figure 5.3.

Listing 5.1 represents the Verilog-AMS non-linear description of a diode. This description contains the name of the module, the contribution statement (comprises non-linear equations), disciplines (electrical), parameters, and branches. The other two components, the resistor, and the voltage source are connected with the diode, so two more linear equations are present. Listing 5.2 is the PWL model of the diode.The model presented in the Listing 5.2 is simulated by the top-level presented in the Listing 5.3 that stimulates the model. After applying all the steps of the methodology detailed in Figure 5.3, through the C++ back-end, the final simulable code is generated. The final generated C++ code is represented in the Listing 5.5, while the analog branch structure is shown in the Listing 5.4. Moreover, this final C++ code is simulable through a simple C++ main code (see Listing 5.6.

Simulation results

The models previously described are simulated with a PC equipped with an i7-9700 processor with 3.00 GHz, 16 Gb of RAM DDR4, and Ubuntu 20.04.2 LTS. A comparison of different simulations of the same model described at different abstraction levels and simulated with different modes is presented. For all the models, two different simulations are performed:

- The non-linear Verilog-AMS model simulated through a Simulation Program with Integrated Circuit Emphasis (SPICE)-based simulator;
- The abstracted C++ model is built starting from a PWL description written in Verilog-AMS.

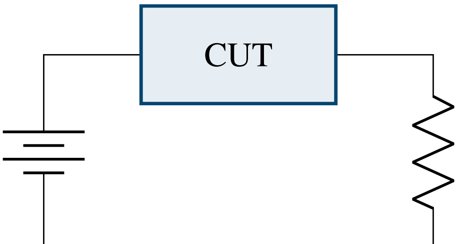


Fig. 5.5: Circuit simulation setup.

Both the simulations are run for 1 *ms* with a fixed time-step set to 1 *us*. Figure 5.4 presents the simulation traces of the half-wave rectifier simulated with a SPICE-based simulator and with the proposed C++ model. The SPICE traces are obtained with the simulation of the non-linear Verilog-AMS model simulated with a SPICE-based simulator. While the C++ traces are obtained with the simulation of the PWL model abstracted and simulated in C++. Inside the graph, the V(input) waveform is the voltage input of the model (a *sin* wave), the V(output) waveform is the output voltage of the system, and the V(diode) waveform is the voltage on the diode. The condition that forces the switch between the forward bias and the reverse bias in the diode is set to 0.69629 V (built-in junction potential (V)). The simulation traces of the two models are equivalent, with an error around 0.1%. This implies that the abstracted model is fully equivalent with respect to the original non-linear model. Moreover, the simulation of C++ code has a speed-up around 2.65x in comparison with the simulation of Verilog-AMS code with a SPICE-based simulator. This speed-up obtained is not related only to the level of abstraction of the C++ language but also to how the methodology for the abstraction of PWL models is built. The methodology exploits the concept of creating before the simulation all the possible solved system of equations that allows describing all the model behaviors. Section 5.1.4 shows the simulation results for the half-wave rectifier, the memristor, and the ideal relay. The simulation time obtained by simulating the heterogeneous model (Verilog-AMS) and the abstracted model (C++) is presented for each model.

Table 5.1: Simulation results for the half-wave rectifier, the memristor, and the ideal relay.

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

5.1.5 Integration into a Virtual Platform (VP)

The half-wave rectifier presented in Section 5.1.4 is embedded into a reference VP described in Figure 5.6. The rectifier is connected to the VP to the MEMS to provide the current DC voltage to guarantee the correct functionality of the MEMS. This VP is composed of an Advanced Microcontroller Bus Architecture (AMBA) bus that interconnects all the components of the VP internally. The components connected to the AMBA bus inside the VP are a com6502 microprocessor, SRAM, UART, a radio-frequency transceiver, and a Micro Electro Mechanical Systems (MEMS) accelerometer composed of Analog to Digital Converters (ADCs), Digital to Analog converters (DACs), and our half-wave rectifier to provide the correct DC voltage to the MEMS. The advantage of integrating several heterogeneous components within the same VP is that a validation of the functionality of an entire system can be performed before taking it to the production stage [218]. In this way, it is possible to obtain an efficient functional simulation using the C++ language. Moreover, it is possible to reduce simulation times and modify the various designs quickly, if necessary, to obtain a complete system in which the functionality is validated. Furthermore, when the system is complete, it is also possible to inject faults into the digital and analog parts to verify the robustness of the designs created to verify functional safety.

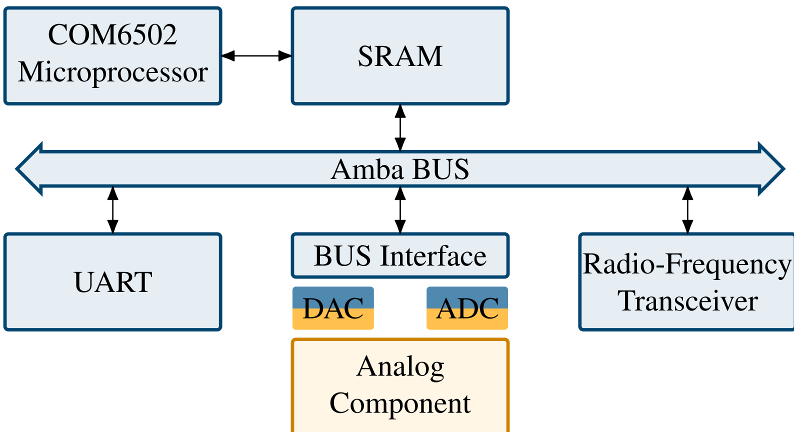


Fig. 5.6: Heterogeneous virtual platform.

5.2 Abstraction of non-linear models

This section proposes a general flow to abstract the behavior of transistor-level circuits (both linear and non-linear) to Reduced Order Models (ROM) that combines a Verilog-A and C++ code. The entire methodology is envisioned and developed in collaboration with the *Sydelity B.V.* company [1]. Thus, all the contents presented in this section are strictly confidential. Figure 5.7 presents a general overview of the flow used to abstract transistor-level circuits to behavioral-level models described in detail in Section 5.2.2 Furthermore, the final model implementation can be written in several target languages: Verilog-A, VHDL-AMS, and Modelica. Thus the approach is not limited to transistor-level circuits but could be extended to other physical domains, e.g., mechanical.

5.2.1 Behavioral modeling

Behavioral modeling techniques can be subdivided into two groups: top-down or bottom-up (data-driven) behavioral modeling. The main strengths and limitations of the top-down approaches are:

[1] https://www.sydelity.com/

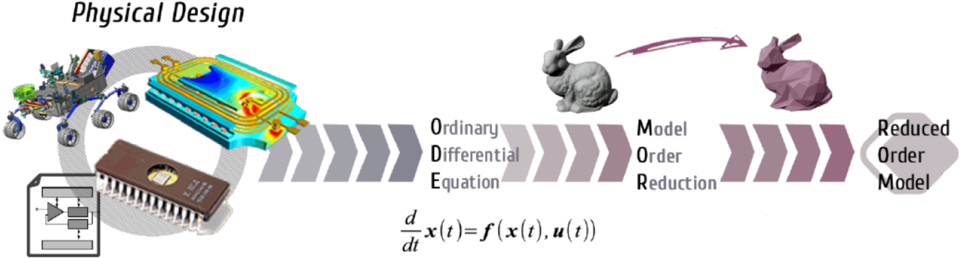


Fig. 5.7: Overview of abstraction of non-linear models to behavioral model.

- Captures the behavior in equations;
- Template-based or handcrafted;
- Parameter tuning with an optimizer or by hand;
- Focussed on the main signal paths;
- Time-consuming modeling process;
- Risk of convergence issues;
- Faster simulation.

While the main strengths and limitations of the bottom-up approaches are:

- Captures the behavior from data (data-driven): data retrieved from simulations, processed, and mapped into Ordinary Differential Equation (ODE).
- Pin-compatible for drop-in replacement of the models;
- Adjustable speed/accuracy trade-off;
- Automatic generation process;
- Simulation does not have convergence problems;
- Faster simulation.

Let us analyze the main publications on data-driven behavioral modeling, summarized in Table 5.2. The commonalities amongst these state-of-the-art methods are related to the collection of small-signal data

Table 5.2: Main articles related to data-driven behavioral modeling.

Criterion	Bradde [219]	Tarraf [220]	De Jonghe [221]
Source Simulator	CDS Spectre	CDS Spectre	MGC Eldo
Data Collection	AC transfer functions @ OP	MNA Matrices @ OP	MNA Matrices @ OP
Data Processing	Vector Fitting	PZ Extraction	Vector Fitting
Analog state	Continuous	Discrete (LTI regions)	Continuous
Replay Engine	Polynomial Interpolation	Piecewise (polyhedra)	AI generated Expressions
Target Simulator / HDL	LTSpice and C++	Verilog-A	Eldo w. VHDL AMS and C++
Accuracy (NRMSE)	Good	Medium Glitches @ boundary X ings	Good
Speed-up	50 -100x	30-120x	10-110x

at Operating Point (OP), implementation of the core as a set of non-linear ODE, and building the static non-linearity using function approximation techniques. In detail, the collection of small-signal data at OP implies:

- Implies the existence of facilities to define the transfer functions that are measured (extraction of Y, Z, H, or S- parameters);

- Highlights the point that DC OP may not be sufficient;
- A lot of data is typically generated in this manner and needs an efficient data format with fast access API.

Then, the second commonality between these techniques is the implementation of the core as a set of non-linear time-invariant ODE as presented in Equation (5.1) and Equation (5.2).

$$\frac{d}{dt}x(t) = F(x(t), u(t)) \text{ with } x(t_0) = x_0 \tag{5.1}$$

$$y(t) = G(x(t), u(t)) \tag{5.2}$$

- ODE correspond to the problem SPICE-like engines are designed to solve;
- The non-linearity is static (it has no explicit time-dependency);
- Requires support for writing ODE in the analog Hardware Description Language (HDL) languages (e.g., indirect assignment in Verilog-A).

The last commonality between these techniques is the building of the non-linearity using function approximation techniques:

- None of the cited works uses the built-in table-lookup features;
- All of them implement a specif function approximation method in C/C++;
- Analog HDL should have a foreign language interface allowing access to functions compiled as a shared library (*.so or *.dll).

The proposed approach (see Figure 5.7) is a data-driven method built to be a completely automatic flow.

5.2.2 Abstraction flow

This section describes in detail the parts that compose the abstraction flow, depicted in Figure 5.8.

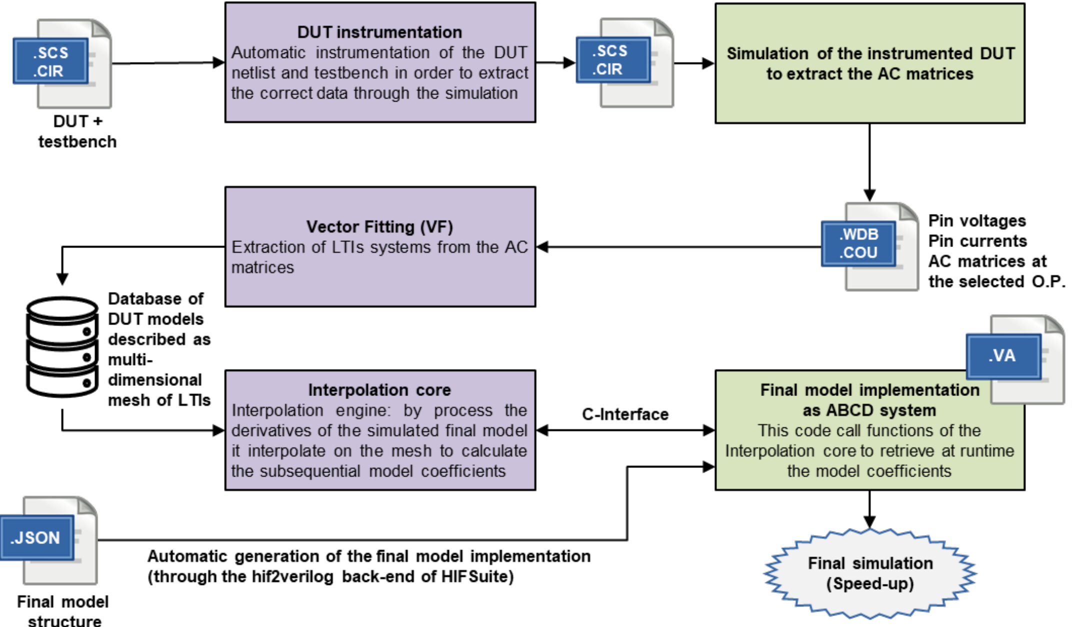


Fig. 5.8: Abstraction flow of non-linear models to behavioral model.

- Initially, the Device Under Test (DUT) that need to be abstracted to the behavioral level is parsed through the front-end of the EDACurry framework (see Section 3.3) that currently supports SPICE, Eldo and Spectre netlist.
- Inside the EDACurry framework, the first main phase of the abstraction flow is performed, the *DUT instrumentation*. This phase requires adding one probe for each port of the circuit and the simulation commands to the testbench. The probes used to retrieve the small-signal matrices from the DUT are S-parameters probes. At the same time, specific simulation commands need to be added to the testbench in order to perform one transient simulation of the DUT and at specific OPs AC simulations by sweeping the frequency parameter. The distribution of the OPs and density depends on the intrinsic behavior of the circuit and are used to linearize the transient behavior in different points.
- After the instrumentation phase through the EDACurry framework, by exploiting the framework back-ends, it is possible to produce in output SPICE, Eldo, or Spectre netlist instrumented with the necessary ports and commands.
- Then, a simulation of the instrument netlist through a global transient simulation and one AC analysis for each OP is performed through SPICE-based simulators, e.g., Eldo or Spectre. The simulation will produce a database (*.wdb or *.cou) that contains the AC matrices resulting from the simulation of the DUT for each OP.
- The AC matrices resulting from the simulation of the instrumented DUT are used as input for the second main phase of the abstraction flow, the *Vector Fitting (VF)*. The VF algorithm is used to create an ABCD system for each OP.
- After the VF phase if the original model is linear only one Linear Time-Invariant (LTI) system is produced through the VF algorithm. Instead, if the original model is non-linear, multiple LTI systems are produced through the VF. In this second case, the set of LTI systems is organized in a multi-dimensional mesh, in which the number of dimensions depends on the input of the DUT. If the initial model is linear, then the interpolation phase is not required because it is sufficient to automatically produce the final model implementation without the need to define external C-functions because all the coefficients of the ABCD system are static. Instead, if the original model is non-linear, the interpolation phase is required.
- The third main phase of the abstraction procedure is the *interpolation core*, which reads the multi-dimensional mesh of LTIs and performs the interpolation between the different system responses. At runtime, based on the input voltages of the circuit passed through the C-interface, the correct LTI system is selected to compute the model coefficients for each simulation step.
- The penultimate step of the methodology is related to the automatic generation of the final model implementation that contains the calls to the interpolation core. This final model could be written in different languages: VHDL-AMS, Verilog-AMS, Modelica, and C++, as it describes a standard ABCD model. To be able to run the final model properly, the simulator chosen needs to support the exchange of the derivatives. In our case, the final model implementation is written in Verilog-A and automatically generated through the `hif2verilog` back-end of HIFSuite (see Section 4.1.7 for more details). This final model implementation is a Verilog-A code that defines an ABCD system with derivatives and external functions to the interpolation core used to get the model coefficients (residues).
- The last phase of the abstraction methodology is the simulation of the final model implementation through SPICE-based simulators. The simulation of the final model runs faster with respect to the simulation of the original model, enabling the integration of this ROM into VP.

The libraries written to implement this abstraction flow, which allows performing the DUT instrumentation, VF, and interpolation core, are coded in C++ and accessible to dedicated python wrappers to automatize the abstraction flow. Now, the three main phases of the abstraction flow, *DUT instrumentation*, *VF*, and *interpolation core*, are discussed in detail.

Device Under Test (DUT) instrumentation

The *DUT instrumentation* is the first main phase of the abstraction flow (see Figure 5.8). The instrumentation is performed by using the developed EDACurry framework presented in Section 3.3. The framework reads in input the DUT descriptions and the testbench written in SPICE, Eldo, or Spectre. After the parsing phase, the DUT represented internally to the EDACurry framework as an AST can be manipulated easily through pre-defined functions. The manipulations performed by the EDACurry framework are: injection of probes for each port of the DUT, adding OP definitions on the testbench, and modifying the simulation commands in the testbench in order to perform one transient simulation and one AC analysis for each OP. For example, Figure 5.9 shows a DUT with two ports instrumented with probes for each port. The injection of the probes, if required, can be performed internally to the dut in order to preserve the external connections with other design modules.

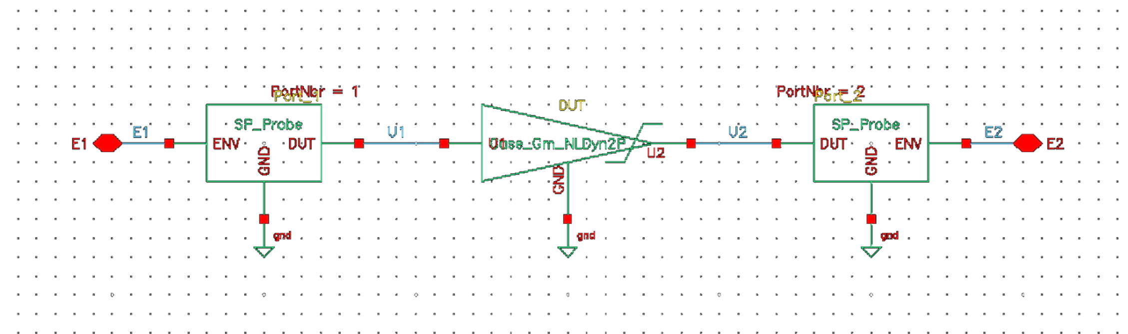


Fig. 5.9: Schematic of a DUT with two ports with probes.

Vector Fitting (VF)

The *VF* is the second main phase of the abstraction flow (see Figure 5.8). This adapted algorithm is used to extract from the AC matrices (retrieved through the simulation of the instrumented DUT) a set of LTI systems that depend on the number of OP used to linearize the time domain simulation. VF is a robust numerical method for rational approximation in the frequency domain using poles and residues [222]. It allows to calculate multi-port models directly from measured or computed frequency responses. The resulting approximation has guaranteed stable poles that are real or come in complex conjugate pairs, and the model can be directly converted into a state-space model [223].

VF is widely applied in many engineering societies, from high-voltage power systems to microwave systems and high-speed electronics. The application is typically to calculate a reduced-order passive macro model from characterizing terminal frequency responses, also known as black-box modeling [224]. The proposed implementation of the *VF* respects the same technical characteristics showed in [219].

Interpolation core

The *interpolation core* is the third main phase of abstraction flow (see Figure 5.8) and have as main objectives:

- retrieves nearest neighbors during the simulation, based on the internal state of the model;
- compute linearized responses based on each of them;
- interpolate the responses to ensure smooth transitions.

As shown in Figure 5.10 the LTI systems produced through the *VF* phase are then organized into a multi-dimensional mesh in order to apply the interpolation between the different LTI systems [225]. The interpolation is guided through the model parameters exchange through the C-interface at runtime during the simulation of the final model implementation.

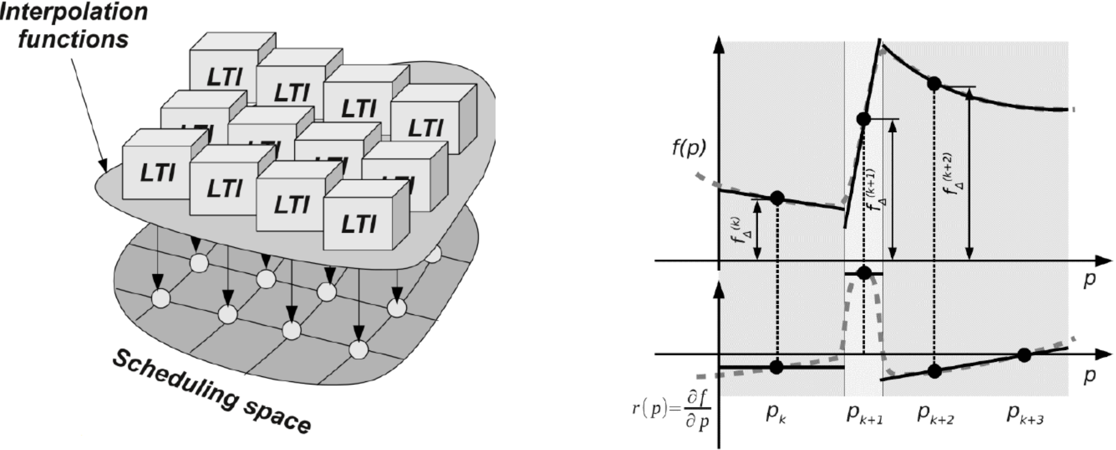


Fig. 5.10: Interpolation between the local responses.

5.2.3 Verilog-A module example

This section shows the implementation in Verilog-A of a simple Wiener-type non-linear system shown in Figure 5.11, demonstrating the feasibility of representing such kinds of models in Verilog-A by exploiting the indirect contribution statements. At first, there is a linear filter, followed by the non-linearity. `H1` is in the example the hyperbolic tangent (tanh()) function.

The implementation of this simple Wiener-type NL (see [226] for details on Wiener-type NL) system is described in Verilog-A by exploiting the indirect contribution statement shown in Listing 5.7.

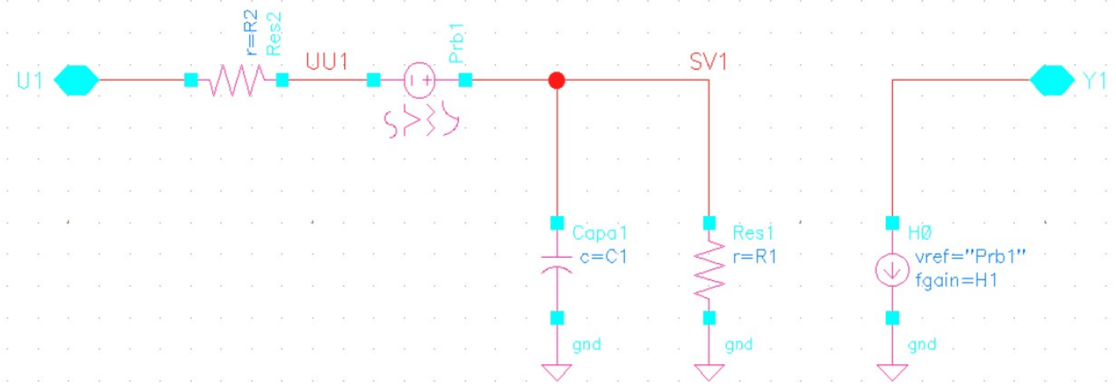


Fig. 5.11: Schematic of a simple Wiener-type NL system.

```
1  // module to demonstrate a simple ABCDE implementation
2  `include "disciplines.vams"
3  `include "constants.vams"
4
5  module abcde(u1, y1);
6      inout u1, y1;
7      electrical x1, x2, u1, y1;
8
9      parameter real R1 = 1e6;
10     parameter real R2 = 1e3;
11     parameter real C1 = 1e-9;
12     parameter real L2 = 1e-15;
13     parameter real H1 = 1;
14     parameter real Y1 = 1;
15
16     real A11, A21, A22, B11, B21, C11, C12, C21, C22, D11, D12, D21, E11, E22;
17
18     analog initial begin
19         // Compute the ABCDE coefficients
20         // There is one single internal state-variable and one input
21         E11 = -1.0*C1;
22         A11 = (R1+R2)/(R1*R2);
23         B11 = -1.0/R2;
24         E22 = L2;
25         A22 = -1.0;
26         A21 = -1.0/R2;
27         B21 = 1.0/R2;
28
29         C12 = 1.0;
30         D12 = 0.0;
31         C22 = -1.0*Y1;
32     end
33
34     analog begin
35         // Equation for the state variable
36         V(x1): ddt(V(x1)) == (A11*V(x1) + B11*V(u1)/E11;
37         V(x2): ddt(V(x2)) == (A22*V(x2) + A21*V(x1) + B21*V(u1)/E22;
38
39         // Equation for the output (with a non-linear saturation effect on y1)
40         I(u1) <+ C12*V(x2);
41         I(y1) <+ C22*tanh(V(x2)*H1/Y1);
42     end
43 endmodule
```

Listing 5.7: Implementation of a simple Wiener-type NL system in Verilog-A.

6

Application to Industrial Cyber-Physical Systems

In the last decade, technological advancements, both in terms of networking, electronics, and information systems, deeply affected the manufacturing environment at all levels: both existing production lines and under-construction ones can now consider opportunities that might have been marginal until now [227]. Every industry must thus adapt to external changes involving people, the global economy, and the environment: it must react quickly to survive in the extreme globalization context while at the same time reducing costs and downtime to the minimum. Information that comes from the production line about its real-time evolution can be used to support the decision-making process and to react to external changes at every level effectively [228]. This chapter proposes some solutions to *help the digital transformation of a production line* throughout its life span, from when it is designed and built to when it is reconfigured to produce different targets. A production line can be seen as a composition of Industrial Cyber-Physical System (ICPS) interconnected, each other with different physical connections (e.g., mechanical, electrical, Radio Frequency (RF)).

In a production plant, data must be efficiently stored and processed to ensure its availability and usefulness. This requires the construction of a *data collection and management architecture* that involves data gathering from sensors placed on the line, its storage according to a precise organization (i.e., local or cloud-based), and its pre-processing and cleaning, so to allow an effective analysis to do what-if analysis, research, development, and process optimization. Finally, the collected data can be exploited to *monitor the operation of the production line* through the construction of a *digital twin* that predicts plant behavior at run time and detects any unexpected discrepancy to activate maintenance and monitoring actions to early identify any malfunctioning and reduce both downtime and costs [20]. In this perspective, monitoring the extra-functional properties (e.g., vibrations, temperature, power consumption, *etc*) of the plant is a very crucial aspect of the production line since it is related to costs and waste of natural resources and can give insights on possible malfunctioning of machinery [229]. In this context, the digital twin of an ICPS, or a composition of ICPS, enables many applications. If faults are added to a ICPS by exploiting the methodologies proposed in Chapter 3 and Chapter 4, a fault-based analysis could be performed for different machine's operating conditions to assess the functional safety in the entire production plant. This chapter is organized as follows:

- Section 6.1 proposes different methodologies to build an ICPS model, and to add faulty behaviors;
- Section 6.2 describes different techniques that can be applied to monitor the status of an ICPS;
- Section 6.3 presents a data collection and management architecture that allows monitoring an entire production plant;
- Section 6.4 proposes a methodology that allows creating a digital twin for an ICPS that enables predictive maintenance strategies.

6.1 Modeling Industrial Cyber-Physical Systems

A smart manufacturing system must integrate a very heterogeneous set of components and simultaneously preserve the functional safety of the whole production line by maintaining a resilient production. In this context, it is strategic to build a hierarchical digital twin of a production line as a composition of ICPS models. Digital twins are born to support the complete digitalization of a smart factory by extending the traditional capabilities through a *virtual replica*. Moreover, by extending the models inside the twin with different classes of faults, it is possible to assess and manage the functional safety of the whole system. This assessment is not limited to each part, but it is computed globally for the overall production line (see Figure 6.1).

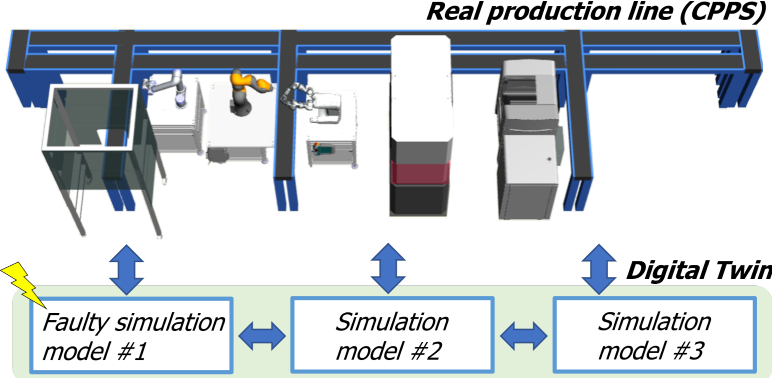


Fig. 6.1: Challenges on building and simulating a digital twin for a production line (composition of ICPS) that integrate faults. What are the best solutions?

The main challenges are:

- choosing the suitable languages/tools to build and simulate the models;
- finding the appropriate framework to simulate models that come from different domains, e.g., discrete or continuous;
- assessment of the functional safety of the production plant, or critical ICPS composing a plant.

The proposed state-of-the-art methodologies start from the system model definition by exploiting the System Modeling Language (SysML) language to model a production plant. For example, a production plant is usually composed of a series of heterogeneous ICPS, combining the *cyber* and the *physical* parts. Through the SysML language diagrams, it is also possible to model both the cyber and the physical parts (through the extension of SysML named SysPhS [1]) of a Cyber-Physical System (CPS) [230]. By modeling all the machines that compose a production line with the diagrams of SysML, it is then possible to export them into a series of Functional Mock-up Units (FMUs) that follow the de-facto standard Functional Mock-up Interface (FMI) for the co-simulation of complex systems [231]. For each CPS that composes a production line, it is possible to create different models (different FMUs) that share the same interfaces but with the internal functionality described at different levels of abstraction. In this way, it is possible to create a hierarchical digital twin. The different FMUs that compose the entire production line are selected concerning the simulation constraint by obtaining a hybrid simulation.

Then, verifying the functional safety of the whole production line is possible by simulating the hierarchical digital twin. Moreover, the models are extended with different classes of faults defined for the

[1] https://www.omg.org/spec/SysPhS/

abstraction level considered. Faults are classified and modeled concerning the domain (cyber or physical) [232] and could vary based on the level of abstraction on which the FMUs are modeled. By simulating the different ICPS composing a production line with faults, it is possible to understand possible failures and define strategies to prevent them by following a "how-to" strategy to anticipate disruptive failures. Furthermore, with the proposed hints for constructing a digital twin, it is possible to support the definition of a complete and tested Failure Mode and Effect Analysis (FMEA). This process could also identify possible critical paths that cause failures and enable the designers to design CPS more resilient to faults or predict "what-next" through the simulation, enabling alternative scenarios, robustness analysis, and optimization.

6.1.1 Systems modeling for digital twin construction

Today, engineers must deal with multiple Domain Specification Languages (DSLs) and poorly interoperable tools to model a CPS. The design phase mostly remains a manual process; system-level requirements are often managed by relying on text-based languages, leading to ambiguities and potential misinterpretations. In this context, SysML [233] enables new perspectives in the modeling phases of CPSs [230]. SysML is a standardized general-purpose DSL for Model-Based Systems Engineering (MBSE) [234]. Generally, engineers work on different tools focused on specific physical domains, making the design of a CPS complicated and risking committing several errors due to the different formalisms used. Usually, the functional safety is validated for each CPS in isolation [235].

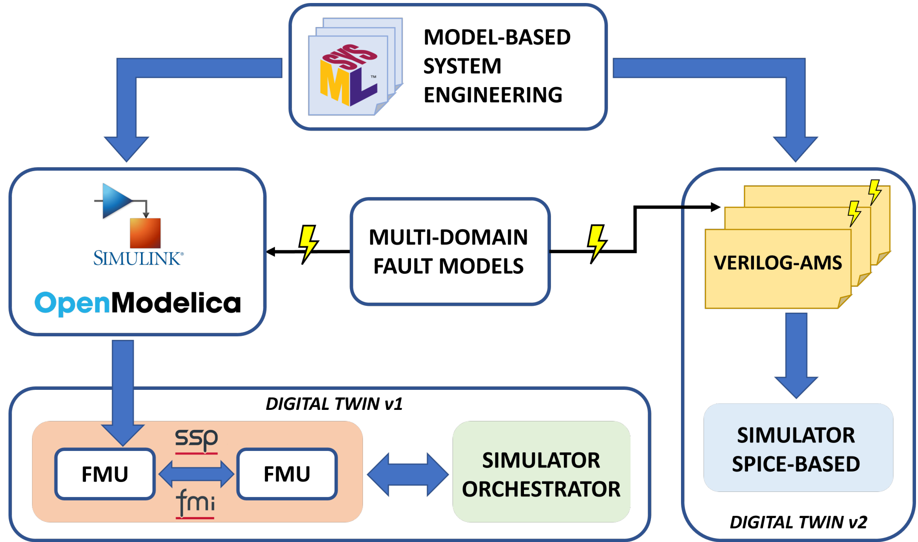


Fig. 6.2: Different strategies to combine digital twins and multi-domain fault models to assess the functional safety of an entire production line.

This section proposes new methodologies and tools for the simulation of a ICPS into a hierarchical digital twin by leveraging the SysML language. To create a digital twin of a ICPS, two strategies have been identified, which are shown in Figure 6.2. Both strategies begin with a description of the model with SysML language: all system properties, interfaces, and functionality are specified. From this description, the system can be implemented at different levels of abstraction. This means that for each CPS, it is possible to describe different models in more or less detail but that maintain the same interface with the outside.

The first approach for building a digital twin (left side of Figure 6.2) starts from the system modeling with SysML, and by using the extension SysPhs [236] it is possible to create direct block diagrams.

These models expressed with block diagrams are coded with the Simulink structure or the Modelica-based structure. Once the system has been modeled in these environments, an FMU containing our system can be generated. An FMU basically is a Zip file containing different information: the ports, binary files, and additional files of the module. Using these approaches allows to simulate the whole production line. Furthermore, it is possible to simulate many CPSs simultaneously thanks to an orchestrator module. The combination of the FMI and the System Structure and Parameterization (SSP) standard allows the creation of this orchestrator module. Regarding the FMI standard, the latest release 3.0 [237] that integrates the newest interface *Scheduled Execution* for the simulation, allows combining in a unique simulation cyber models encapsulated inside an FMU and physical models encapsulated inside an FMU. With this improvement to the standard FMI, more complex CPS can be simulated, enabling the construction of a complex digital twin.

The second approach (right side of Figure 6.2) exploits the features of the Verilog-AMS language to describe the behavior of the CPS for which a digital twin is required. Only the differential equations that identify the behavior of the system are constructed, while the physical details are left out. The starting point for this second approach remains the SysML language. It is possible to generate an XML that contains all the model information, starting from a model described with the diagrams supported by the language. Then, starting from the XML description, it is easy to derive a model written with the Verilog-AMS language. A Verilog-AMS model can be simulated using a transistor-level simulator based on Simulation Program with Integrated Circuit Emphasis (SPICE) language, such as Eldo or Spectre. Furthermore, also the interaction of the cyber with the physical part can be simulated with SPICE-based simulators.

Moreover, with the proposed approaches, the functional safety of the whole production line can be validated by developing specific fault injection techniques for each level and domain of the hierarchical digital twin. The proposal of studying the functional safety of a whole production line is innovative because it would be performed by simulating it with different classes of faults into a *holistic digital twin*. Furthermore, it is possible to calculate a diagnostic coverage more accurately with the proposed methodologies because the hierarchical digital twin allows considering a production line as a unique system composed of many CPS. Instead, the classical approach calculates the diagnostic coverage for each CPS in isolation, without considering the influence that a faulty CPS could cause on the other CPS interconnected with it inside a production line [238].

6.1.2 System-level simulation

Following the methodologies proposed in Section 6.1.1, the models that compose a digital twin can be constructed in several ways. It is possible to identify two macro-categories: the models of the systems could be built with block-based components or through languages, e.g., Verilog-AMS or SystemC-AMS. Once defined, it is necessary to identify the best simulation techniques and tools needed to simulate them. Following the literature, the first simulators developed were those for electrical circuits, SPICE-based. Several classes of simulators were built from these early simulators, focusing their capabilities on simulating models in isolation. While with the advent of Industry 4.0, it became increasingly necessary to find simulation techniques that could relate different models within the same simulation. Furthermore, these models during the various design phases are modeled at different levels of abstraction based on the design analyses performed on them. In Figure 3.12, the graph shows the differences that exist in simulating models described at different levels of abstraction. For example, if we consider very accurate models of electrical circuits, e.g., described with SPICE-based languages, maximum model accuracy will be achieved, but also low simulation performance [239]. Then, analyzing the middle part of the graph, there are conservative models, for example, models described using Modelica or Verilog-AMS language. With conservative models, there is a

balance of accuracy and simulation performance. Also, if more simulation performance is needed, switching to Real Number Model (RNM) could be useful, but the simulation loses accuracy. Another category of emerging models is the Reduced Order Models (ROM), which attempts to mimic the behavior of a system by exploiting various mathematical or Machine Learning (ML) techniques. The latter category of models cited can also be considered part of the RNM within the proposed graph.

In this context, creating a hierarchical digital twin capable of integrating models from different physical domains and with models described at different simulation levels is critical. Depending on the analysis to be performed through the digital twin, models described with the required level of abstraction will be instantiated and simulated. It is necessary to anticipate that multiple models described at divine levels of abstraction exist of the same CPS because the computational complexity of simulating a digital twin is often onerous. Therefore simulating multiple described CPSs with the highest accuracy would be impossible, so an attempt is made to select and simulate models according to the required analysis of the digital twin. For example, a common problem could be to see how a subcomponent is stressed by the components of the environment of which it is a part. In this case, the most detailed model for that section of the system has to be chosen besides the abstract models for the components of the surrounding environment. This approach can be implemented by taking advantage of the capabilities of languages such as Verilog-AMS or System-C AMS or any solution that support the FMI standard. This standard allows unique interfaces to be defined for heterogeneous models, so linking multiple models is possible. The SSP standard ensures the composition of different FMUs within a single simulation design, which is capable of defining the interactions between different FMUs using an XML description. When it is necessary to simulate heterogeneous components, it is required to consider how to make the cyber part interact with the physical part of a CPS. Leveraging the proposed flows for building a digital twin makes this possible: the Verilog-AMS language natively supports control statements to interface and synchronize a digital simulation with an analog one. This strategy allows the simulation of digital control of a physical component, e.g., a Direct Current (DC) motor. Similarly, exploiting the FMI 3.0 standard and the Scheduled Execution simulation interface is possible to interface cyber models with physical models encapsulated within FMUs. Finally, it is crucial to consider the architecture on which a digital twin has to be simulated: it could be locally or in a distributed environment. If the computation is local, such as on a personal computer or a server, it is necessary to reserve the appropriate quantity of resources for computation. Whereas, if the computation is too onerous or the system under analysis is composed of multiple CPSs, the possibility of moving the computation to the cloud should be considered. The Distributed Co-Simulation Protocol (DCP) standard has been developed to distribute the computation of several FMUs that need to interact with each other, with a view to an increasingly complex and evolving digital twin.

6.1.3 Faults modeling

In the context of a production line, simulating becomes fundamental in order to ensure functional safety in the working environment [9]. A complete system operation flow can be obtained by combining real operation data and simulated data traces. Therefore, it is possible to detect when and how a given fault ruins the functionality of the system by comparing a normal run with a failed one. These faulty execution data, relative to extreme cases of operation or dangers to the system itself, must be acquired by the simulation to avoid unnecessary maintenance costs or machinery replacement costs. In order to perform these operations, a fault taxonomy is needed to specify which faults effectively affect the system behavior [8].

Fault modeling in different physical domains

Faults represent wrong behaviors of a system that can happen for multiple reasons, like device aging, the breaking of an internal component, a digital failure, or a production defect. Functional safety studies how it is possible to improve the security of a device by using adequate security systems. An industrial environment usually consists of many components belonging to different physical domains, including electrical and mechanical ones. For each of these domains, several fault models are defined by the cause, how they occur, and what effect they have on component behavior. Indeed, the same fault in a component can cause different overall effects, depending on the production line structure or its setting environment.

In the electrical domain, there is a well-defined classification of faults at the behavioral level [240]: they are injected into the differential equations describing the model, changing its functionality. In an electrical circuit, a fault is a *saboteur* if it consists of a new component that alters circuit behavior by injecting a new set of equations describing the faulty behavior. The fault is a *mutant* when it is a mutation of an existing component, e.g., a change in the nominal value of a parameter or a modification that models an abnormal behavior in the presence of faults. In contrast to the mechanical domain, several fault models are part of the state of practice in the electrical world.

Since there are different levels of abstraction in the electrical world, other classes of faults also exist. Different fault classes can be applied depending on the level of abstraction of the electrical circuit model. If the model is described at the transistor level, the faults to be injected belong to this class, such as stuck-on and stuck-off defects [32]. On the other hand, if the circuit is modeled digitally, faults will occur at the logic level, such as stuck-at 0 or stuck-at 1 [241, 242]. Similarly to the electrical domain, also for the other domains of a production line, there are fault models not only at the equation level but also at the physical level. For example, in the mechanical domain, several taxonomies involving materials, geometric components, shapes, and sizes are listed in [146, 151].

Regarding Figure 6.2, faults are injected differently in each system modeling and simulation methodologies. An exemplification of the different classes of faults applied to a CPS is shown in Figure 6.3. However, the fault models injected into the systems equations are the same for the two descriptions since they are both described at the behavioral level. Instead, fault injection techniques change depending on the level of abstraction of the model and the language with which it is expressed.

Failure Mode and Effect Analysis (FMEA)

In the context of the analysis and injection of mechanical failures, the FMEA cannot be omitted, which is often a core task in reliability engineering, safety engineering, and quality engineering [243]. This methodology is used to analyze the failure or defect modes of a process, product, or system; it analyzes its causes and evaluates what the effects on the entire system. Generally, the analysis is carried out in advance and is therefore based on theoretical and not experimental considerations.

The process consists of reviewing as many components, assemblies, and subsystems as possible to identify potential failure modes in a system and their causes and effects. For each component, fault modes and their effects on the rest of the system are recorded in a specific FMEA worksheet. A successful FMEA activity helps identify potential failure modes based on experience with similar products and processes or common failure logic physics. It is widely used in the development and manufacturing industries in various product life cycle phases. Effects analysis refers to studying the consequences of those failures on different system levels [244].

So, in detail, the procedure always acts on the device under test in optimal conditions:

- one fault at a time;

- the component is powered normally and receives correct commands;
- all consumable resources are present in sufficient quantity.

The analysis starts by listing the functions the design needs to fulfill since a design is meant to be the only possible solution to perform functions that need to be satisfied. Thus, faults are studied as variations of these functions, which describe the system through some parameters: the probability of occurring fault, the severity of the issue, and whether and how the failure can be detected. These parameters identify the risk level of the problem, which can range from low to unacceptable. Other useful variables are brought back from the analysis of a single fault: the potential causes, the effect of the failure at low and system level, and how to repair it.

However, only a few parts of the FMEA realization has been automated, for example, the symptoms generation [245]. Furthermore, the FMEA worksheet is hard to produce, hard to understand and read, as well as hard to maintain. However, the procedures presented in Figure 6.2 are designed to perform a complete and automatic FMEA analysis. The goal is very ambitious, but by implementing this flow, it would be possible to obtain results comparable to this type of technique.

6.1.4 Coupling digital twins with faults

In this section, the two methodologies presented in Figure 6.2 are applied to a simple CPS to compare them. The two proposed methodologies differ in the modeling phase and in the software used for the simulation. The system considered is a DC motor, modeled in both Simulink (see Figure 6.4) and Verilog-AMS (see Listing 6.1). In the Simulink version, we can see the block description, composed of the electrical part (blue) and the mechanical part (green). Instead, the version of the DC motor in Verilog-AMS is realized via differential equations and specifying connections between branches. Each corresponds to a different equation, which is connected with the others through variables. Specifically, the branches determine how the effort (voltage for the electrical domain and torque for the mechanical domain) and flow (electric current and angular velocity) variables are connected between the system components.

The DC motor is a multi-domain CPS composed of an electrical and a mechanical part: it converts the direct current into mechanical rotation energy. The electrical part is modeled by a resistor and inductance that model the internal characteristics of the motor coils and the rotor. An electromotive force is connected in series since the conversion between electrical and mechanical energy occurs at this point. The mechanical rotational part models a shaft connected to the rotor with its inertia and friction.

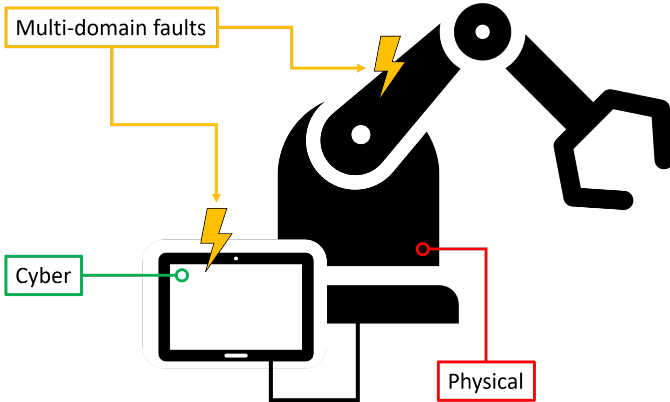


Fig. 6.3: Multi-domain fault injection into a CPS (e.g., an anthropomorphic manipulator with its digital control).

```
`include "disciplines.vams"
`include "constants.vams"
`timescale 1us / 1us

module motor(shaft, p, n);
    // PARAMETERS -----------------------
    // Motor constant (V-s/rad)
    parameter real km = 4.5;
    // Flux constant (N-m/A) = Kt
    parameter real kf = 4.5;
    // Inertia of the shaft (N-m-s2/rad)
    parameter real js = 1.2;
    // Drag rotating friction of the shaft (N-m-s/rad)
    parameter real ds = 0.1;
    // Motor winding resistance (Ohms)
    parameter real r = 5.0;
    // Motor winding inductance (H)
    parameter real l = 0.02;
    // PORTS --------------------------
    output shaft;
    input p, n;
    // NODES --------------------------
    electrical p, n;
    // Internal nodes.
    electrical n1, n2;
    rotational_omega shaft, rgnd;
    // Reference nodes.
    ground rgnd;
    // BRANCHES -----------------------
    branch (p, n1) Vm;
    branch (n1, n2) R1;
    branch (n2, n) L1;
    branch (shaft, rgnd) bshaft;
    // BEHAVIOR -----------------------
    analog begin
        // Electrical model of the motor winding.
        V(Vm) <+ km * Omega(bshaft);
        V(R1) <+ r * I(R1);
        V(L1) <+ l * ddt(I(L1));
        // Physical model of the shaft.
        Tau(bshaft) <+ + kf * I(Vm);
        Tau(bshaft) <+ - ds * Omega(bshaft);
        Tau(bshaft) <+ - js * ddt(Omega(bshaft));
    end
endmodule
```

Listing 6.1: Original DC motor model, fault-free.

Table 6.1 shows the DC motor simulation time data implemented following the two methodologies in Figure 6.2. All the simulations are run on a workstation with Windows 10, 16 gigabytes of RAM, and a Core i7-9700 run at 3.0 GHz. In particular, the simulators have been set using the same simulation parameters: 10^{-3} seconds is the fixed simulation time step. The DC motor described in Simulink/Simscape was exported as FMU for co-simulation 2.0. The time reported in the second column of the table refers to the simulation of the FMU using Simulink [2]. Then, the second column reports the times of the simulation of the

[2] https://it.mathworks.com/products/simulink.html

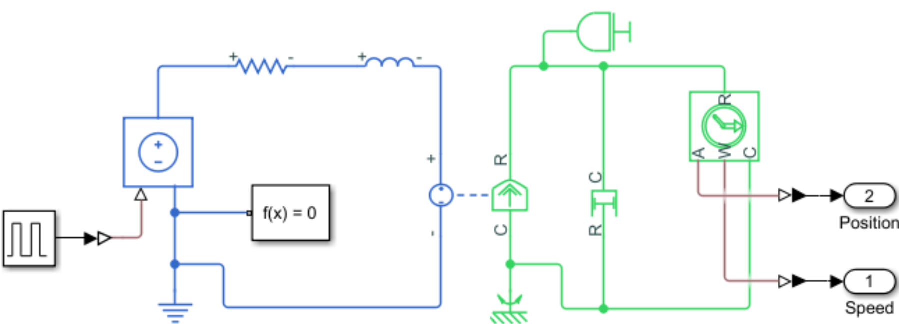


Fig. 6.4: DC motor plus inertia modeled in Simulink/Simscape.

Table 6.1: Simulation time measurements.

Simulated Time	Simulink FMU	FMPy FMU	Verilog-AMS
2 sec	8.230 sec	6.274 sec	0.44 sec
4 sec	12.99 sec	11.196 sec	0.76 sec
8 sec	22.68 sec	13.805 sec	1.15 sec
10 sec	27.91 sec	15.197 sec	1.61 sec
20 sec	54.63 sec	27.391 sec	1.95 sec
30 sec	77.69 sec	39.123 sec	2.24 sec

FMU using the open-source FMPy library [3]. Finally, in the fourth column, the simulation time of the model described in Verilog-AMS and simulated with a SPICE-based simulator. It is evident from the results that the SPICE-based simulation is faster, while the FMU-based simulations are more limited. A model encapsulated within an FMU (for co-simulation) must manage the interaction between the internal FMU solver and the simulator solver. In contrast, only the simulator solver simulates the Verilog-AMS model. Moreover, Verilog-AMS allows a combined and efficient simulation of the cyber and physical parts of a CPS, enabling complex interactions between the cyber and the physical parts. Starting from the initial modeling of the system through SysML (see Figure 6.2), one of the two methodologies has to be chosen: through Simulink or Modelica equivalently, or the Verilog-AMS language (or VHDL-AMS or SystemC-AMS) is up to the designer. The results that can be obtained from these two flows are comparable.

Finally, recall that both proposed flows are designed to model digital twins at the behavior level. Consequently, the faults that can be injected into these models are those suitable at the behavioral level. Regarding Figure 6.2, the faults that are injected are the same in the two proposed modeling methodologies, but the injection techniques differ. Specifically, the fault models (behavioral level) are injected directly into the system equations in a Verilog-AMS model. Usually, a new branch is added in series or in parallel to the branch, i.e., to the equation, to be faulty. In a block environment such as Simulink, faults are injected through real additional blocks that alter the behavior of the system. By simulating the digital twin with different classes of faults, it is possible to generate faulty simulation traces [246]. These simulation traces can be used for different purposes, e.g., to train predictive maintenance algorithms. So, maintaining the digital twin updated and capable of preventing anomalies through the data produced from a digital twin is the main goal of this work. In order to do that, after every maintenance task (e.g., after changing some parts and plant shut off) on the real production line, the digital twin must adjust its parameters based on the data retrieved from the production line. With these adjustments, after each maintenance task, the digital twin can evolve by following the real operating conditions, avoiding producing outlier data.

6.2 Smart monitoring of Industrial Cyber-Physical Systems

Manufacturing over the past decade has invested massive budgets in the digitalization of factory equipment. Each production cell has been equipped with a plethora of heterogeneous Industrial Internet of Things (IIOT) sensors collecting different information, e.g., energy consumption, movement-related, temperature [247]. Such data is typically collected and analyzed with Artificial Intelligence, and in particular with ML approaches, to extract relevant information and patterns, with the goal of ensuring lifelong process

[3] https://github.com/CATIA-Systems/FMPy

monitoring, identifying and reacting to anomalous behaviors and unexpected conditions, and reducing machine downtime [248]. Nonetheless, the application of ML techniques requires a time-consuming manual annotation of time series to distinguish correct from anomalous behaviors and to relate measured quantities with information about production parameters [249, 250]. This manual step introduces many potential pitfalls, as labels may be incorrect or misaligned in time or applied to too few samples to allow effective ML application.

This section proposes an important step forward in data management to go beyond the aforementioned limitations to the application of ML techniques. The key idea is the development of a *Data Integration HUB (DIH)* (see 6.16) that *automatically aligns sensor time series and correlates them with information about the production process*, extracted from the Manufacturing Execution System (MES). At the state-of-the-art, the problem related to the unavailability of the labeled datasets is addressed by using unsupervised algorithms [251]. In our scenario, the focus is on the data fusion of different sources of data plus production process information to monitor industrial equipment. Consequently, by exploiting this proposed data fusion architecture is possible to implement predictive maintenance strategies by combining sensor data and the production schedule of an entire day or week, allowing to predict in advance machines' ruptures. The MES software manages the production plant and exchange information with the Enterprise resource planning (ERP) software. This software is part of the automation pyramid that categorizes all the software layers composing a factory (see Figure 6.7). This creates an extension of the MES, called *Automation Controller*, that allows enhanced data-driven management of the production line.

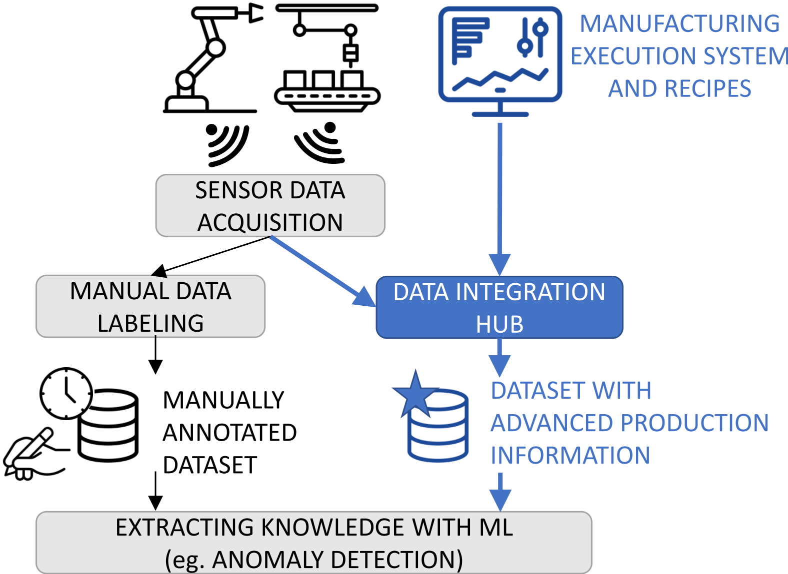


Fig. 6.5: Overview of the proposed data fusion infrastructure that fuses sensor data with recipe data extracted from the MES inside the DIH module, avoiding time-consuming and error-prone manual annotation of the dataset, allowing enhanced extraction of knowledge with ML.

To achieve effective data fusion, this work brings the following contributions (in Section 6.2.2):

1. data originated by sensors at different sampling frequencies are merged by exploiting a global timestep, thus achieving homogeneous data traces;
2. data is then automatically labeled with information extracted from the MES describing machines state, i.e., performed actions (e.g., pick) and relative parameters (e.g., speed and object weight);

3. To avoid manual intervention and potential errors, both processes are automatized, which also makes it easier to produce extensive datasets.

The generated datasets allow the application of ML algorithms to extract relevant information. This is exemplified in the experimental results as follows:

- application of classification techniques to derive machine state (i.e., operation and parameters) from the sensed data to detect anomalous situations (intuitively, if a robot arm is stuck, its sensors will reproduce an idle behavior even if the MES sent a pick request);
- application of refined anomaly detection strategies that operate in a non-supervised fashion, e.g., by identifying unexpected behaviors through the confidence of machine states classification output;
- support to the design phases of the IIOT infrastructure by identifying the optimal transmission frequency and distinguishing between low impacting data traces and sensors that retrieve the most relevant information for the proposed ML algorithms.

6.2.1 State of the art and definitions

This section gives an overview of industrial data collection techniques and some concepts of machine learning applications suitable to be applied to a smart factory, with a comparison of state-of-the-art solutions.

Industrial data collection

Modern equipment produces an enormous amount of data collected with different communication protocols. Among these, OPC Unified Architecture (OPC UA) is a platform-independent protocol based on client/server structure. Each server exposes its information to the clients through a hierarchical, object-oriented structure named the *information model*. It plays a dominant role, becoming the standard de facto for Machine to Machine (M2M) communication in industrial automation. Another popular protocol is MQTT, standardized in IEC 20922. MQTT is a publish/subscriber protocol that runs on TCP/IP, designed for an environment with constrained communication band or computational resources.

Publish-subscriber is an interesting data distribution architecture where a series of entities owning information (publishers) send it to those who want it (subscribers) through a middle-man (broker). A message broker is software that implements the Message Oriented Middleware (MOM) communication paradigm. It provides services in the form of APIs, where publishers can send messages to one or more subscribers defining routing and delivery policies. The two strengths of the MOM paradigm consist of: (1) messages are placed inside queues for which persistence is guaranteed, and (2) it guarantees that the order of the messages is preserved from the subscribers' perspective.

In a scenario with several pieces of equipment producing large volumes of data, which need to arrive at several destinations, publish-subscriber architectures can guarantee scalability, fault tolerance, and low-latency communication. Examples of distributed publish-subscriber messaging systems are Apache Kafka and RabbitMQ [252]: Kafka is suited for environments where low latency and throughput are primary concerns, while RabbitMQ offers higher security and request-response messages and is based on the Advanced Message Queuing Protocol (AMQP). Once brokers collect the data, the following steps concern cleaning, filtering, and structuring such data through a process called *data fusion* [253]. A critical aspect of all data fusion approaches is labeling of the data: labeling is oftentimes done manually, which proves to be complex, inefficient, and prone to errors [254]. Once you have robust and reliable data collection/fusion architecture in place, the next step is applying ML techniques that can support data-driven decision-making [255].

Machine learning applications into a smart factory

ML techniques are widely adopted for dealing with a variety of production line problems, as they are very effective in analyzing complex systems and addressing challenging issues [248]. Two common ML tasks are:

- detecting product failure through *classification* (maps input features to one output variable or label) or *anomaly detection* (identifies outliers w.r.t. the normal operation of the production line) [256];
- reducing the number of input features through *data reduction*, which filters noisy or repetitive data points and highly correlated or irrelevant features.

One of the main challenges of applying state-of-the-art ML techniques to such tasks is the unavailability of labeled data to properly train the models [257]: in a standard industrial setting, human intervention is often crucial to annotate the data as well as to select features, algorithms, and their respective parameters. Results obtained with these techniques depend on the quality of the information that can be extracted from the collected data [258]. Therefore, methodologies enabling advanced data collection, such as real-time data labeling, are necessary to improve the quality of the produced dataset.

6.2.2 Data fusion in Industry 4.0

This section details the behavior of the DIH to handle the data fusion proposed in this work. It takes in input (Figure 6.6):

- *data measured by sensors* positioned on/in the production equipment, here as an example, gyroscopes and accelerometers to describe movement (numbered) and a power sensor (P); data traces may be at different time scales (left, Section 6.2.2);
- *commands originated by the MES* according to production recipes (task graph and verbose form) with the corresponding operating parameters (right, Section 6.2.2);

and generates a sound data architecture where all traces are aligned, and MES operations are used to label data (below, Section 6.2.2). The final architecture is explained in Section 6.2.2.

Communication with IIoT sensors

Generally, different IIOT sensors are positioned directly on different machines composing the shop floor in a production plant, and they communicate through wireless or wired network solutions. Such sensors generally belong to different classes allowing for the detection of extra-functional properties of the monitored machines and environmental data (e.g., vibration, temperature, power, acoustic emissions).

The DIH infrastructure proposed in Figure 6.6 exploits two brokers, Kafka and RabbitMQ. Brokers allow more flexibility in creating a data collection infrastructure as they enable the connection of heterogeneous machines and sensors that support different industrial communication protocols. Kafka guarantees throughput and low-latency communication between publisher and subscriber. Therefore, all the machine and sensor data are published to Kafka. It also allows sending data in batches reducing the computational effort, particularly for embedded devices. Constrained sensors with minimal computational resources publish messages to MQTT. Finally, industrial machine services, which enclose machine functionalities, are exposed through OPC UA. All such services are connected to the DIH through RabbitMQ Remote Procedure Call (RPC) queues.

Distributed architectures do not have a shared clock and, therefore, a global timestamp. By listening to its published messages, each sensor and machine compute a Round Trip Time (RTT) value between itself

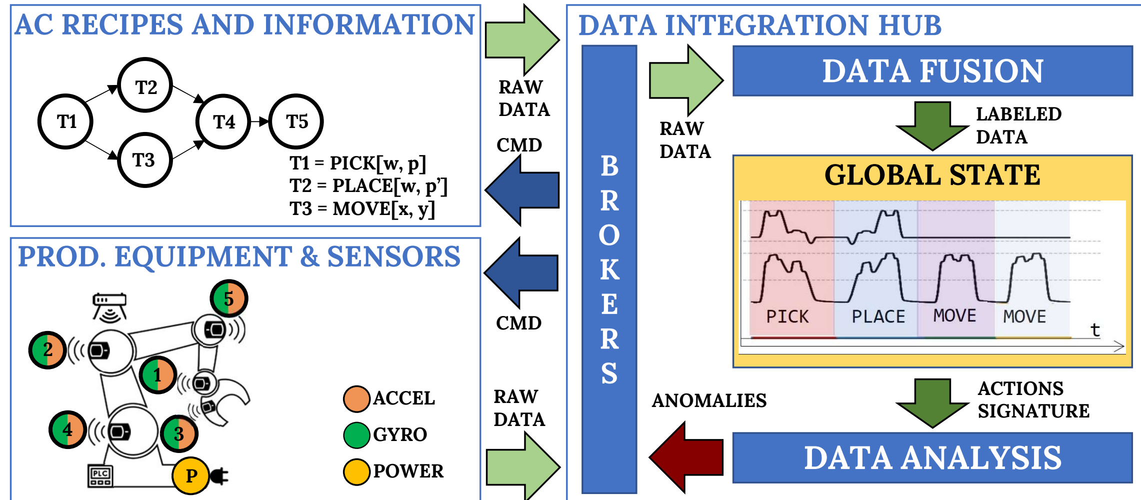


Fig. 6.6: Role of the DIH: it takes in inputs data coming from sensors (potentially at different frequencies, here gyroscopes, accelerometers, and power sensors, left bottom) and commands originating from the Automation Controller (with relative parameters, left top) and fuses all the data traces by automatically annotate them with production information.

and the destination broker. Brokers define messages' global timestamp based on their receiving time minus the RTT value. The global timestamp associated with the proposed infrastructure is thus an approximation that allows for aligning different data together. In addition, the proposed infrastructure enables bidirectional communication, so machines and sensors can expose services/commands implemented by other modules in the DIH. This enables the automatic setup and reconfiguration of sensors to increase or decrease the frequency of sending data on demand. For example, we can avoid over-saturating the bandwidth by reducing the frequency with which sensors of *idle* machines send data (we cannot avoid sending data altogether because we might need historical (even partial) data for running safety-critical analyses, e.g., identifying the series of events that lead to irregular behaviors). In addition, the proposed infrastructure allows sensors to be added and removed according to different monitoring needs through automatic plug-and-play configuration. Finally, all the different modules that make up the DIH are encapsulated in Kubernetes containers, making the infrastructure even more flexible and scalable.

Communication with MES

The communication between the MES and the DIH is realized through a custom *software component* called Automation Controller, which extends the MES functionalities. The Automation Controller software is positioned at the same level as the SCADA/HMI as depicted in Figure 6.7. The complete architecture of the Automation Controller is described in [259]. Modern MESs allow interaction with other components through *interfaces* that support different communication protocols. This allows for adding custom logic such as scheduling policies, automatic execution, and automatic feedback from the production plant. The Automation Controller is connected to the DIH through Kafka for receiving data streams and through RabbitMQ to send machine commands. Meanwhile, the communication with the MES consists of an RPC client calling its functions.

Through the Automation Controller, we have extended the MES scheduling policy, implementing a fully-automatic execution of production recipes. In addition, our components expose the MES functions to all the actors of the DIH, enabling bidirectional communication. When the Automation Controller calls

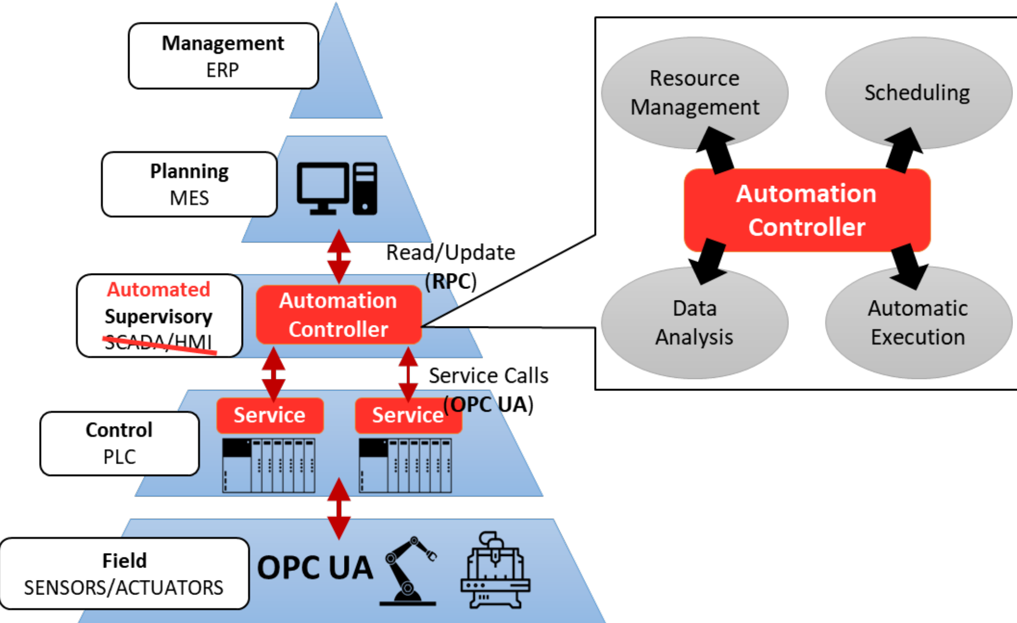


Fig. 6.7: Automation pyramid: subdivision of each architectural level inside a production plant. The Automation Controller is positioned at the same level as the SCADA/HMI, and work as a supervisor that automatizes the production.

a machine command, it publishes it with its parameters and *state* to the DIH, as depicted in Figure 6.6. Specifically, we have defined these possible states as follows:

- `Init`: when the Automation Controller has sent the command to the machines, but the machine response is unknown;
- `Failed`: if the machine rejects the received command;
- `Start`: corresponds to a positive response for the received command;
- `End`: sent by the machine, it identifies the end of the received command.

These states represent the *evolution* of a command, seen from the Automation Controller perspective. Moreover, it is possible to fuse this information with data from different sources, such as sensors and machines, by automatically labeling the data. All the actors of the DIH can then use these data to implement additional functions such as Anomaly Detection.

Data Integration Hub (DIH)

Inside the DIH, data are correlated using the global timestamp related to each packet that encapsulates data. The DIH fuses commands from the Automation Controller with data of machines and sensors as shown in Figure 6.6. It does so by extracting information regarding plant topology from the Automation Controller. In particular, it extracts the *hierarchical structure* of a production plant, working site, working area, and working unit. This allows for identifying which sensors are related to a specific piece of equipment and labeling data only with commands of the same piece of equipment. Data and commands are fused together by exploiting the global timestamp of the `Init` and `End` state. These two states identify the boundaries in which data are associated with a command. This granularity allows associating sensor time series with a small overhead of idle time at the start and at the end, caused by communication delay (i.e., RTT). The data fusion procedure constructs a single *row* for each piece of equipment containing the global timestamp, the current command with its state, and all sensor values. Each row represents a *sample* of machine state. Thus, sampling at a specific frequency avoids synchronization problems between sensors that send data at

different frequencies. The automatic labeling of the data allows precise temporal correlation with the phases of each machinery, avoiding manual transcriptions.

Case study

The proposed infrastructure enabling data fusion has been implemented within the Industrial Computer Engineering laboratory part of the Computer Science department of the University of Verona [4]. It consists of a full-fledged production line in which all different types of industrial processing are present, e.g., additive manufacturing, subtractive manufacturing, collaborative robots, and different types of material transport between different processing stations. The whole architecture is applicable to different industrial machinery and is exemplified here on a collaborative anthropomorphic manipulator, a Kuka Lightweight Robot (LR), with seven degrees of freedom. Several sensors were placed on this manipulator and connected to the DIH through different industrial protocols (top left of Figure 6.6). The sensors considered for monitoring the Kuka manipulator in this work are 5 accelerometers (for acceleration), 5 gyroscopes (for orientation), and a three-phase energy meter that allows power monitoring. The sensors provide 56 raw signals simultaneously varying over time in a multivariate time series scenario. The accelerometers send data via MQTT, while the energy meter sends data directly to Kafka. In addition, the manipulator has an OPC UA server that allows it to send commands received from the MES directly via the RabbitMQ broker. The commands identify 31 different desired actions to be performed by the robot, which translate into corresponding categorical labels.

Dataset

The data management applied by the DIH allows to automatically tag the data with the information coming from the MES. The resulting data is generated at entries that correspond to the overall "state" of all the sensors in the plant and current production and can be sampled at different frequencies (it can vary depending on the data rates of the sensors considered). In addition, however, it is possible to do this in real-time within the DIH by having the data tagged to Kafka as rows of the dataset are sent individually. Figure 6.8 exemplifies this by showing examples of the raw signals provided by the sensors, where the colored horizontal bar indicates the corresponding action (made of action and parameters) that the robot is doing at the given time frame.

6.2.3 Extract knowledge with ML

To demonstrate the potentialities of our framework, we apply a set of state-of-the-art ML algorithms to the Kuka robot arm and analyze the results in a number of relevant tasks.

The initial dataset used for the application of ML techniques is the result of 3 hours of data collection. The obtained dataset (raw sensors signal plus associated command) enables the automatic prediction of the robot's actions, hereafter referred to as machine states classification, where the command is the ground-truth class label. The resulting ML approaches will then be activated at run time during the operation of the Kuka's arm.

From sensor to features

The first stage of a ML pipeline is the *Feature Extraction*, which in our case, implies transforming the raw sensor signals into a compact set of features to be fed into the machine states classifier. As visually represented in Figure 6.9, each raw signal is appropriately windowed over time according to the corresponding

[4] https://www.icelab.di.univr.it/

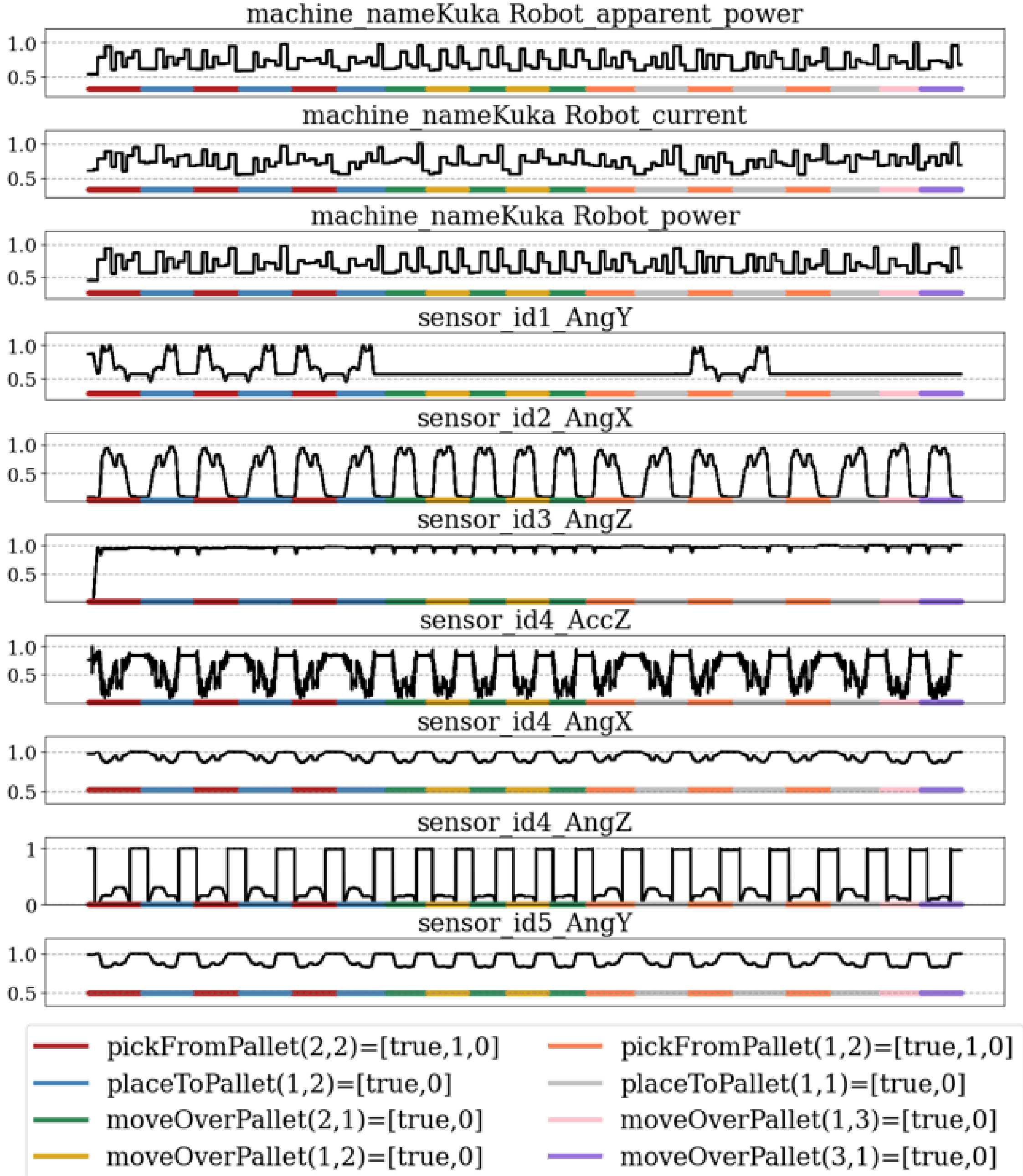


Fig. 6.8: Representative examples of the raw signals provided by the sensors. The horizontal bar indicates the univocal *action-parameters* tuple of the Kuka at a given time frame. Note that the set of the univocal tuples *action-parameter* establishes the desired categorization for the 31 machine's states.

action of the robot. Then, we extract a set of 1980 statistical features exploiting the TSFEL [260] Python package.

To ease the following classification process, the extracted features undergo two typical pre-processing steps:

1. *Feature standardization*, consisting in removing the mean and scaling to unit variance the original feature values.

2. *Feature selection* with Minimum Redundancy Maximum Relevance (mRMR) algorithm [261] to find an optimal set of features that is mutually and maximally dissimilar and can represent the target class effectively. This reduces the features set from 1980 to 50.

Machine states classification

In our experiments, we put into effect five different state-of-the-art classifiers: RF, Support Vector Machine (SVM), AdaBoost (AB), Multilayer Perceptron (MLP) and Bayesian Multilayer Perceptron (MLP-B). The scikit-learn [262] Python library was used to implement, train and optimize RF, SVM, AB. MLP and MLP-B were built on top of the Keras framework [263]. Hyper-parameters optimization is performed through randomized search [264]. To assess the robustness of the classifiers, training, and validation, implement a standard 5-fold cross-validation strategy. The available dataset is split into 5 complementary subsets, each containing 20% of the data. At each iteration, a classifier is trained on 80% of the data and tested on the remaining 20%. The cross-validation accuracy is obtained by averaging the accuracy rate at each iteration.

Figure 6.10 provides the mean cross-validation accuracy of the classifiers at different sampling rates of the sensed data (respectively: 1, 10, 100, 200 Hz). The error bars in the graph indicate the standard deviation of the accuracy among the different classes (that is: the shorter the bars, the more balanced the classification accuracy). As expected, all the classifiers have increasing accuracy values at increasing sampling frequency. MLP is the one that provides the highest and most balanced mean accuracy values (about $97.6 \pm 2.5\%$ at 10 Hz). Interestingly, the accuracy curve presents a clear elbow point at 10 Hz, which provides a very useful indication of the best compromise between sampling rate and classification performance: at this sampling rate, most of the tested classifiers have mean accuracy above 80%. Given the obtained results, we will focus the downstream analysis on raw signals always sampled at 10 Hz.

The proposed architecture can work with good accuracy even with low-frequency data; this implies two main advantages. First, it can be applied to older production lines where the pre-existing sensors work at lower frequencies. This is still the case in most realities because several Small and Medium-sized Enterprises (SMEs) are still in the midst of the digitalization process and are not equipped with new technologies yet. Second, most industrial networks host moderate traffic for production control. As such, high-frequency data sent by multiple sensors might hinder the normal and safety-critical operations of the production line. As proved by this experiment, the construction of a dataset of sensor data annotated with information extracted from the MES constructs a solid base for the ML algorithms.

From features to sensor

A more in-depth analysis of the classification results can provide better insights into the relevance of the sensed data to machine state prediction. It is possible to understand which sensors and/or specific physical

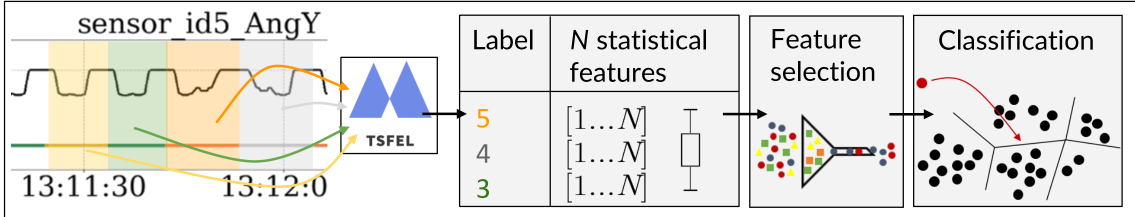


Fig. 6.9: Overview of the ML pipeline. The raw signal (e.g., *sensor_id5_AngY*) is windowed based on the robot's actions and associated with corresponding labels. Then, it is fed into the TSFEL library to extract statistical features, which are reduced to a 50-dimensional set by feature selection. The final feature set is fed into a classifier.

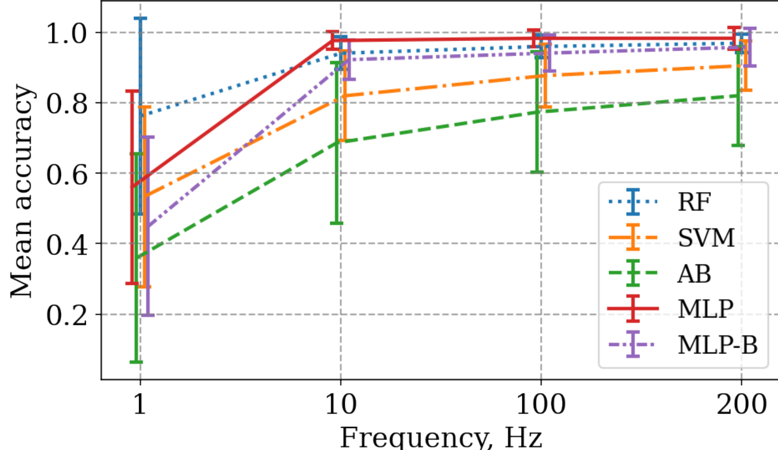


Fig. 6.10: Mean cross-validation accuracy of different machine state classifiers at different sampling rates of the sensed data. Error bars indicate the standard deviation of accuracy among different classes.

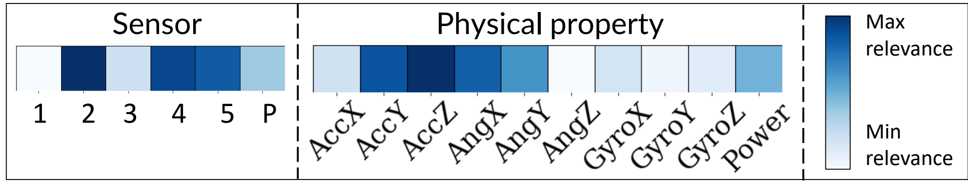


Fig. 6.11: Sensors/physical properties color-mapped by their relevance. For sensor positioning, refer to the top-left section of Figure 6.6.

properties most affected the classifier decisions. This is important not only as a sanity check but also to guide the optimization of the sensing infrastructure set-up (e.g., by removing/replacing irrelevant sensors). We adopt the following strategy to circumvent the intrinsic *black-box* nature of ML techniques. We start from the top 50 features obtained after the *Feature selection* step, and we reconstruct the association to their raw input signal and the corresponding sensor and physical property. Thus, we group feature occurrences by the sensor/physical property. The ratio is: the higher the occurrence of features derived from a given sensor/physical property, the higher its relevance for the machine states classification task.

The obtained results, after scaling, are represented in Figure 6.11, where darker color means higher features occurrence (and hence, higher relevance) of the corresponding sensor/physical property. As it can be observed, accelerometers 2, 4, and 5 (whose position is visible in Figure 6.6) provide the most part of the information to the classifier, while the most relevant physical properties appear to be the acceleration and angle of the machine arms. Vice versa, accelerometer 1, as well as all gyroscopes data, seems to have a much lower relevance to the task. As anticipated, this gives relevant feedback for the construction of the production line, as sensors may be configured to extract only relevant data, thus reducing bandwidth occupation, and less relevant sensors can be removed and installed on other parts of the line. While the specific outcome of this analysis is not generalizable because it is tied to the classification task and commands that were analyzed, it indeed provides a relevant example of how the data fusion infrastructure can be exploited with ML.

Anomaly detection

Early detection of anomalous behaviors of the production line is crucial to avoid severe deterioration in productivity and prevent dangerous accidents [265]. Nonetheless, anomaly detection is still a challenging

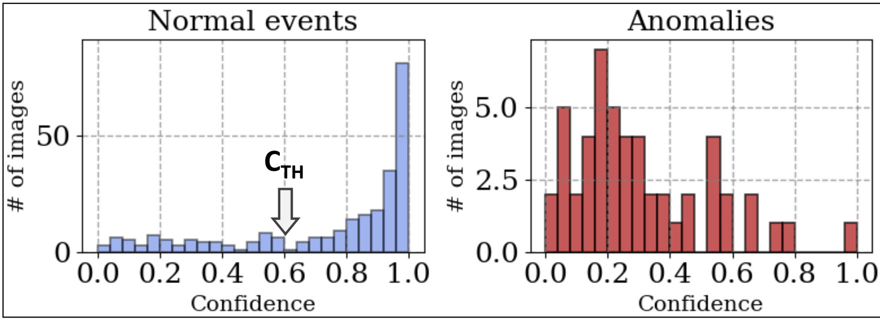


Fig. 6.12: Distribution of the confidence values (MSR) of MLP-B predictions for the normal events (left) and for the anomalies (right). The threshold C_{TH} is indicated by an arrow.

open problem because anomaly data records are (i) rare; (ii) irregular; (iii) typically lacking prior information, and (iv) hardly predictable. In this regard, training a supervised ML algorithm to recognize anomalies requires a huge amount of training data, with many pre-annotated anomalies as the target classes. Such a dataset is extremely cumbersome to collect and more often unfeasible in the early life of the production line when the equipment is still behaving correctly most of the time [265]. On top of these considerations, in our work, we choose a different approach, which does not require specific supervision of the anomalies. We start from the machine state classifier described in the previous sections, which is built on top of data without anomalies. Then, we exploit this model to identify anomalous behaviors at inference time indirectly. Such a learning paradigm, where a supervised model learns a *pretext task* (in this case, machine states classification) to solve a different *downstream task*, is generally known in the literature as Self-Supervised Learning (SSL) [266]. In our case, anomalies can be indirectly identified in two scenarios:

1. the action predicted by the classifier is different from the one specified by the MES;
2. the predicted action is correct, but the decision has relatively low confidence. This is reasonable to assume, given that the classifier was asked to perform a prediction on data that are *anomalous*, thus significantly different from the ones it was trained on.

While the first scenario is straightforward, the second requires a statistically sound measure of prediction confidence, which is generally not provided by deterministic ML models. For this purpose, we exploit MLP-B, a probabilistic implementation of MLP based on Bayesian theory. Differently from conventional training methods that attempt to find a single set of optimal values for the neural network weights, the Bayesian approach estimates the posterior distribution of the weights and makes a prediction by integrating over this distribution [267]. To produce a distribution of weights (and hence, of predictions), in our work, we use the Monte Carlo Dropout method, in the popular implementation by [267]: random dropouts (e.g., random switching-off of neurons) are used to produce many different networks at inference time, that can be treated as Monte Carlo samples from the space of all available models. This allows inferring a distribution of predictions per each given input, providing the mathematical grounds to estimate a confidence level: the so-called Maximum Softmax Response (MSR) (i.e., the maximum value of Softmax activations in the last MLP layer) [268].

To assess the goodness of our approach, we extend the available dataset, already described in the previous sections, by artificially adding 56 anomalies created by manually hampering the robot motion at random instants during data acquisition. The so-obtained dataset is fed into the MLP-B, and the distribution of MSR confidence values obtained during machine states classification are shown in Figure 6.12, separately for the normal events (on the left) and the anomalies (on the right). From these plots, we can see how MLP-B confidence is very high with *normal* events (peak of distribution in the left plot is at MSR=1.0) and significantly

drops in the presence of anomalies (peak of the right plot is at MSR=0.2). This demonstrates the validity of our initial hypothesis that the classifier confidence can be indirectly used to identify anomalies.

Starting from these considerations, it is reasonable to assume that the overall distribution of the predictive confidence is bi-modal, where the highest mode is associated with high-confidence values, which are the majority, and the second to the low-confidence ones (e.g., anomalies). Hence, to automatically identify an anomaly, we just need to establish a threshold C_{TH} on the confidence level. The most straightforward way to do so is to compute the value that splits the confidence values of the training dataset into two groups with maximum inter-group variance (see the arrow in Figure 6.12). By applying this approach to our dataset with artificial anomalies, again with a standard 5-fold cross-validation strategy, we obtain that anomalous events are recognized with a mean accuracy of 79% (sensitivity 84%, specificity 75%). These results are remarkable for two reasons: i) the anomaly detection is fairly accurate, even without having received specific training on the anomalies to be detected, which would be difficult to implement in real industrial applications; ii) the training only relies on the data that is automatically collected, integrated and annotated during a normal production process. This is a further demonstration of the usefulness of our proposed data fusion infrastructure.

6.3 Smart monitoring of a production line

The concept of *Industry 4.0* originates from the will to introduce the benefits of digital computation into new and existing industrial plants to save time, materials, and energy. Digital transformation requires that all the machinery of a production line is connected together with the enterprise applications to capture and analyze data across all manufacturing stages. Then, collected data can be exploited to make strategic decisions

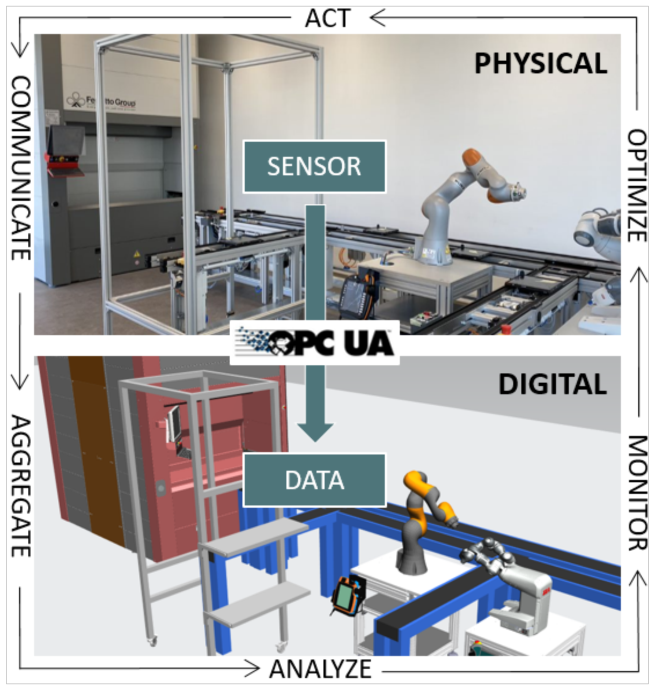


Fig. 6.13: ICELab digital twin.

on the production and to monitor it, reacting to unexpected behaviors and thus reducing downtime and

maintenance costs. Figure 6.13 presents an overview of the methodology flow for monitoring a production line. This work aims to show the key elements of such digital transformation by exemplifying them in the Industrial Computer Engineering (ICE) laboratory of the University of Verona. The ICE laboratory is a production line developed for research and technology transfer of concepts related to the new industrial revolution. First of all, this section presents a methodology to build a digital twin of the production line, monitoring its behavior and, in particular, its power consumption. Any discrepancy between real-time data and those taken from the evolution of the digital twin allows triggering an early intervention on the line to guarantee effective maintenance. By analyzing the data flowing over the network infrastructure, any anomalies in production can be identified, for example, by analyzing energy consumption data.

6.3.1 Data collection and management

Achieving communication with the physical plant is a key step in its digital transformation and opens the door to smart manufacturing [269].

Data sources

In manufacturing, relevant data comes both from the operations management infrastructure (Figure 6.7) and from sensors installed in the production environment: the former provides information about the operation of the line (e.g., the production recipe being executed, status and configuration information about the equipment), while the latter allows monitoring the physical evolution of the machinery (e.g., power consumption, vibrations, movements) [270]. The correlation of these different sources of data provides the necessary knowledge about the evolution of the production line.

The choice of sensors to be installed on the production line is crucial, as they are the real heart of a smart factory. Considering the heterogeneity of the quantities to be monitored, different sensors will be applied to data acquisition. Transversely to all monitored quantities, the choice must take into account the technical characteristics of the sensor in terms of connectivity and of integration with the enterprise software.

Data is then collected in a variety of ways: environmental sensors usually adopt IoT infrastructures, whereby equipment and product information can be collected directly from the operations management infrastructure, e.g., through dedicated servers and database technologies [270, 271]. Different sources imply different data formats and types that must be integrated to allow efficient processing and intelligence extraction. Data formats involving a large number of bits may lead to reduce the sampling rate if the channel capacity is low, thus compromising the real-time effectiveness of data collection.

OPC Unified Architecture (UA)

The aim of the OPC UA communication protocol is to standardize the machine to machine communication [272]. OPC UA is the current OPC Foundation's technology for secure, reliable, and interoperable transport of raw data and pre-processed information from the field level into production planning systems. Through the *OPC-UA Specifications* is started a process of standardization of data exchange between software applications in an industrial environment.

The OPC UA architecture is composed of Clients and Servers as interacting partners. Each Client may interact concurrently with one or more Servers, and each Server may interact concurrently with one or more Clients. OPC UA can be used at different levels of the automation pyramid (see Figure 6.7) for different applications within the same environment. At the plant floor level, an OPC UA server may run in a controller providing data from field devices to OPC UA clients (e.g., HMIs, SCADA). On top of the

plant floor at the operation level, an OPC UA application may be a client collecting data from the server at the lower level, performing special calculations, and generating alarms; an example is represented by an OPC UA client integrated into an ERP system, obtaining information about used devices in the plant floor and creating a maintenance request. Inside an OPC UA server is contained the information model, in which all information regarding a machine is stored. This model is a graph data structure and can be used to represent a wide variety of structured information. Concepts of object-oriented languages, such as type hierarchies and inheritance, are also incorporated into OPC UA. The totality of the data an OPC UA server exposes is therefore called address space. Being a graph, it is made of nodes and edges. Nodes are identified by their NodeId and are divided into NodeClasses. There are eight different kinds of NodeClasses: Objects, Variables, Methods, and View, which are instances of the types ObjectTypes, VariableTypes, and MethodTypes. Edges are called References. Excepting certain nodes that must be present on every OPC UA server, there is no predefined way to arrange the OPC UA address space. This enables the implementation of customs OPC UA server that contains specific address space according to the requirements.

Data storage and processing

A large amount of collected data must be securely stored and effectively integrated. In this sense, *cloud computing* allows to effectively save large quantities of data in a highly cost-effective, energy-efficient, and flexible manner. Additionally, the distributed nature of cloud storage allows highly scalable and shareable storage [273]. This is critical as the stored data includes both real-time monitored data and historical data that must be available to increase awareness of the evolution of the equipment over time and to build prediction and control models [274].

The collected data must then be pre-processed to ease extraction of relevant knowledge: redundant, misleading, duplicated, and inconsistent information must be removed, and data reduction techniques must be applied to transform a massive amount of data into ordered, meaningful and clean information, useful for subsequent analysis [275, 274]. After these steps, data is to be made available to services, such as digital twins, that exploit the real-time line monitoring and data coming from the management infrastructure to improve the production process.

6.3.2 Energy monitoring

Equipment energy consumption is a large portion of the total consumption of manufacturing ($\sim 75\%$), and thus should be tightly monitored and optimized to cope with serious situations such as rising energy prices, global resource depletion, and climate warming [276]. To achieve energy monitoring, the equipment of interest is enriched with sensory devices that measure power or current demand over time. Such data is collected and monitored in real-time, and it can be transmitted over the network to make further analysis about the correlation of such power consumption w.r.t. equipment operation parameters [277, 278, 279].

The construction of the energy monitoring infrastructure requires to intervene both on the physical layer and on the virtual layer:

- identify relevant *machine parameters*, i.e., machine-related information that allows identifying the operation mode of the equipment (e.g., moving, cutting, idle) and the relative configuration (e.g., speed, acceleration) that is considered relevant from the perspective of power consumption;
- insertion of *power consumption sensors* to extract power measurements in real-time during machine operation;
- construction of an *IoT collection and transmission infrastructure* that allows fast collection and integration of data through the adoption of edge technology or of cloud servers for data storage.

Once the real-time data is available, it is compared against *models of power consumption*, either based on historical data or on models built on equipment specifications. This comparison allows the detection of any misalignment of the real equipment w.r.t. the expected behavior as an effect of gradual degradation (e.g., due to wear or corrosion) and sudden disturbances [28]. In this perspective, energy monitoring becomes thus an important instrument to achieve *predictive maintenance*, i.e., localization of quality losses in machining, better prioritization of the maintenance schedule, avoiding unscheduled downtime and losses in product quality [280]. Additionally, the power consumption models allow to *optimize the production process* by comparing the impact of different production scheduling settings or of different parameters of the same production process [28, 281].

A relevant role is thus performed by the chosen energy models that identify relations between machine operation and the corresponding power consumption. Different models have been proposed in the literature, depending on the available data and on the goal of the analysis:

- *power state machines*, a power consumption model inherited by Electrical System Design that identifies the device's typical operation states and associates each state with the corresponding power consumption [282, 283]. The power values are either derived from available documentation or extrapolated from available data sensed from the plant;
- *mathematical and statistical models* describing the manufacturing system as a stochastic dynamic system, characterized by analyzing sensed data [281, 284];
- *learning-based approaches* that exploit actual historical data obtained from the plant to train neural networks or learning algorithms that capture the nonlinear relationships between input parameters of the plant and output power consumption [285, 286, 287].

The kind of approach to be adopted strictly depends on the characteristics of the production line and its data monitoring infrastructure:

- *availability of data sensors*: if data is collected at run time from the plant, it is possible to build data-aware models with a higher level of fidelity w.r.t. plant operation; else, it is necessary to rely on available documentation, thus including an implicit glitch;
- *availability of information related to the equipment*: building a detailed correspondence between device operation and power consumption requires the equipment to export relevant information about its operating parameters, like speed, acceleration, position, *etc.*, without whom it becomes possible to reason only in terms of general idleness or activity;
- *desired level of detail* w.r.t. the actual plant operation, i.e., whether it is necessary to have a high accuracy and adaptability w.r.t. the evolving conditions of the physical plant.

In this way, the power consumption model becomes a *digital twin of the production plant* [20], focusing only on one major aspect, i.e., energy consumption monitoring w.r.t. plant operation. The digital twin is, of course, enabled by a sound data-collection infrastructure that collects real-time data along with historical data to track its energy consumption and by the power estimation models that are used for parameter optimization, scheduling, and equipment upgrading and maintenance.

6.3.3 Case study: Industrial Computer Engineering Laboratory (ICELab)

The reference production line for this work is an Industry 4.0 research facility called Industrial Computer Engineering Laboratory (ICELab). The general structure of the ICELab includes a fully-fledged production line (Figure 6.14):

- a vertical warehouse for storing materials and products;

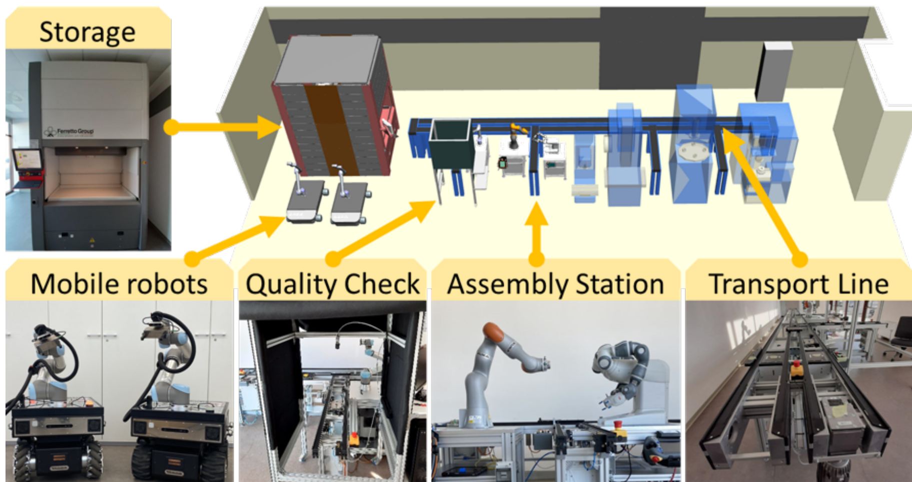


Fig. 6.14: Structure of the ICELab production line.

- two collaborative Autonomous Mobile Robots (AMRs), i.e., two Robotnik RB-Kairos AMR equipped with anthropomorphic manipulators that can load and unload materials from the warehouse to a dedicated point on the conveyor belt actively cooperate with an operator and perform advanced and cooperative handling tasks;
- a quality check station;
- a collaborative robotic assembly station comprising two lightweight collaborative robots: an ABB Yumi and a Kuka Lightweight robot;
- two 3D printers: a stereolithography mono material 3D printer and a multi-material polyjet 3D printer;
- a milling machine;
- an electronic automatic tester;
- a complex transportation system composed of the main conveyor belt that spans across the entire laboratory in a ring configuration and an unloading conveyor bay for each machine and the AMRs.

This particular structure of the laboratory allows for representing the most modern automation technologies adopted in production processes. This work focuses on a subset of the equipment, specifically the parts labeled in Figure 6.14.

The considered production process is divided into four phases. First, a set of LEGO-like blocks is transported from the vertical warehouse to the assembly station by means of the transport belts. Second, the pieces are assembled by the two cooperating robotic arms, i.e., Kuka and ABB. Third, the assembled product is transported to the quality check station by means of the same transportation belts used before. At the quality check station, a robotic arm rotates the assembled piece and exposes all the critical parts to the cameras: if the quality standards are achieved, the piece is then put back on the belt and transported to the vertical warehouse for storage. As depicted in Figure 6.13, all the active entities of the production line are instrumented to provide the IT office with real-time data by using OPC UA protocol with end-to-end encryption [5]. Each sensor implements an OPC UA server that exposes some relevant parameters of the equipment.

6.3.4 Online monitoring and data collection

The ICE Laboratory is innovative because it combines the knowledge of many industrial partners, enabling the sharing of technologies and new ideas in the context of *Industry 4.0*. The combination of IIoT sensors

[5] https://opcfoundation.org/about/opc-technologies/opc-ua/

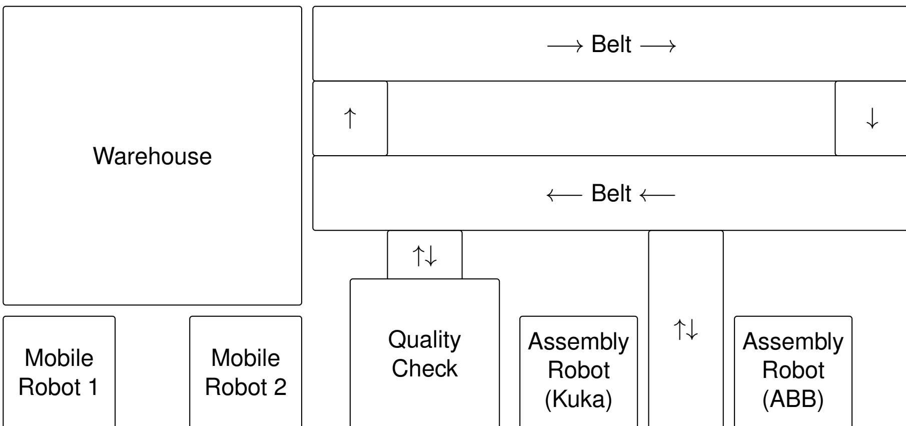


Fig. 6.15: ICELab production line schematical view. Each block composing the figure describes an industrial machinery available in the laboratory. All the machineries composing ICELab are directly connected with the conveyor belt that transports mini-pallets.

and a complete infrastructure for the data collection allows online monitoring of the entire production process [5, 270]. The IIoT sensors placed on the production line are different, e.g., vibration, temperature, position, and power sensors. All these sensors are connected to two BOX-IO gateway [6]. In Figure 6.16, the entire data collection architecture of the ICE Laboratory is shown. The pre-requisite to apply this data collection architecture is a plant equipped with OPC UA servers providing:

- The equipment status by native or custom OPC UA servers;
- Environment data (IoT and Industrial IoT).

An OPC UA server provides secure access to industrial automation data using OPC UA information models. The information model specifies how the data is organized, stored, and collected. The OPC UA server is the software that runs on a machine that provides access to the information and not the hardware itself of an industrial machine. The proposed data collection architecture shown in Figure 6.16 is based on the OPC UA communication protocol that relies on Kubernetes infrastructure [7]. The goals of this infrastructure are:

- Monitoring through OPC UA servers;
- Data logging;
- Data analytics and filtering;
- Data upload to different cloud providers;
- Provide a unique interface to access data;
- Secure connections.

Now, in detail, each block of the data collection architecture (see Figure 6.16) is explained. The sources of the data are the IoT sensors and the equipment (highlighted in yellow). Different OPC UA client reads the data from the sources (OPC UA server application). Each OPC UA client is reconfigurable because it allows to retrieve the data from an OPC UA server through the OPC UA server URI. Then it could subscribe to each variable. For each variable of the OPC UA information model, it is possible to set the sampling interval, the datatype and unit, and other custom static fields. All the data retrieved through the OPC UA clients send the data to a publish/subscribe data buffer (our configuration is based on the Apache Kafka open-source application). This data buffer is a large data stream handling partitioned in different topics. This application is subdivided into multiple instances to guarantee fault tolerance and high performance. Moreover, this

[6] https://edalab.it/box-io-iot-platform/
[7] https://kubernetes.io/

data buffer is extensible because adding producers/consumers of data is easy. The data from the data buffer could safely monitor parts of the production line before storing them in a time-series database. Then, the data needs to be stored in the time series database through data uploader nodes. For each buffer of the data buffer, a data uploader node with a topic group allows scalability. Each node of the data uploader sends chunks of data to the database (our configuration is based on Telegraf, which is a plugin of InfluxDB). Telegraf is a plugin-driven agent that collects, processes, aggregates, and writes metrics into an InfluxDB time-series database.

The time-series database chosen for the ICE Laboratory is a NoSQL database, InfluxDB. InfluxDB is an open-source time-series database that provides real-time visibility into stacks, sensors, and automation data for monitoring metrics and events. The data are stored in a persistent volume and can optionally filter incoming values and perform data aggregation. The data inside the database are organized in buckets. For each data saved in the time-series database, a time-stamp is associated, that is, the time instant in which the data is read from the production line. The best configuration of the database is to subdivide the buckets for different purposes, e.g., raw data, clean data, and cloud data. The database has multiple instances managed by the Kubernetes infrastructure. The data are analyzed through a filter, clean, and manipulation process on the database data. The results of these manipulations are usually stored in a different bucket inside the database. The data collection architecture's final step is the uploading part of the data stored in the time-series database to the cloud providers. Obviously, only the necessary data for other analyses are uploaded to the cloud. The quantity of data on the cloud depends on the company policy. Moreover, through custom applications and viewers, it is possible to visualize the query results on the data stored in the database (our implementation of custom dashboards is based on the Grafana Dashboards). All the nodes of the data collection architecture are bundled in a container. A container is an application abstraction that packages code and dependencies together (application, binaries, and libraries). Each node has been deployed into a Kubernetes cluster. In a physical plant, a system could handle thousands of containers. For that reason, a Kubernetes cluster is required to manage the clusters. Kubernetes is a container orchestration tool from CNCF Foundation and automatically manages the containers without an IT manager's control.

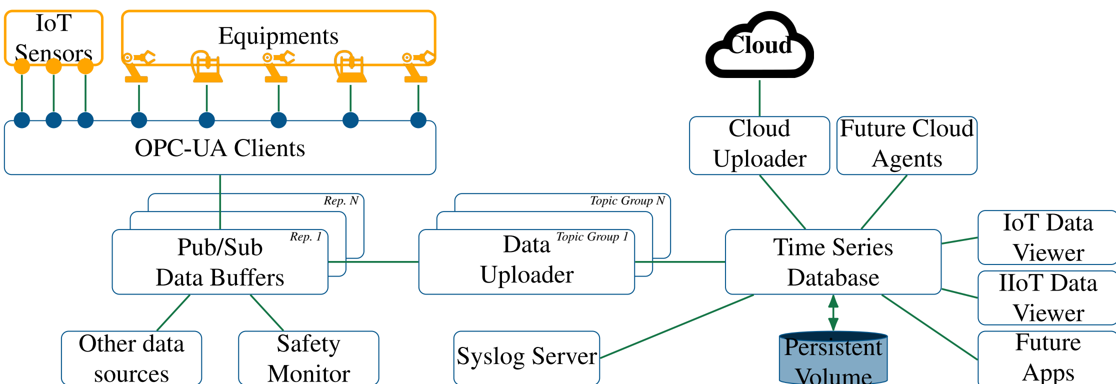


Fig. 6.16: Data collection architecture of the ICE laboratory. Each industrial equipment composing the laboratory has on board an OPC UA server that can be implemented on the computer controlling the machine or into the Programmable Logic Controller (PLC) that support this protocol. The data collection by exploiting an equal number of OPC UA clients with respect to the machinery allows for retrieving all the interesting information from the machinery. This information is processed into specific Pub/Sub brokers enabling to process of large quantities of data. All the data are then saved into a time-series database, enabling to record of all the information with its specific timestamp associated.

In this section, we thus focus on constructing a digital twin for power consumption of the portion of the production line that was equipped with sensors at the time of this experimental analysis: the quality check station, the robotic assembly station, and the transportation system.

Sensing infrastructure

Accessing the power consumption of the devices of interest required the installation of sensors for monitoring the evolution of the AC power quantities [288]: active power (W), phase angle (degrees, i.e., the phase between the voltage and current sinusoidal curves), power factor (used to estimate the amount of dissipated power), and current (A). The ABB Yumi and the Kuka LR are single-phase AC devices, and the quality station is a DC station converted to single-phase AC through an inverter. All such devices are thus monitored through single-phase Eastron SDM 230 power meters [8]. The transport line is made of 15 three-phase AC motors. To ease the monitoring of power consumption, all motors are monitored through a single three-phase Eastron SDM630 meter, exporting the accumulated AC power characteristics of all phases [9]. This sensing strategy allows to consider the transport line as a single-phase AC load, thus avoiding the burden of separating the power models of every single phase. The power sensors allow retrieving the sensed data with a polling rate of 2 seconds in our Modbus configuration.

Data processing

All real-time data, including measurements made by the sensors and information about the production recipe, are made available to the power consumption digital twin through the MES interface (as explained in Section 6.3.4). Information about the device operating mode, exported from the PLC, consists of the macro-state of the device, i.e., whether it is idle or active, and in the latter case, the action performed (e.g., in the case of the robots, pick, or place). More detailed parameters (e.g., operating speed and acceleration) were not available at the time of the experimental analysis. To model the power consumption of the devices, we decided to adopt approaches that exploit both historical and real-time data. This required a collection campaign to store data relative to a number of production cycles and a cleaning phase of such data to extract timestamps, remove duplicated data and identify unavailable samples.

Models of power consumption

The goal of the digital twin is to predict the active power consumption of each device, given information about the production recipe and the device operating mode received at run time: if the corresponding sensed value is different from the predicted one (given a certain tolerance threshold), an alert is notified. Data pre-processing has been applied to the historical data of each device in order to determine the correlation of its settings with the corresponding power consumption of the machine. The developed model will strictly depend on the kind of monitored device and its available information.An analysis of the correlation ρ between the quantities monitored by power sensors proved that all of them are highly correlated with active power: e.g., correlation ρ w.r.t. active power for the Kuka robot is 0.76 for current, 0.89 for power factor and 0.84 for phase angle [10]. Thus, modeling power consumption is a good bias also for the other quantities. This allows for reducing the amount of data that must be transferred from the sensors to the digital twin and

[8] https://www.eastroneurope.com/products/view/sdm230dr

[9] https://www.eastroneurope.com/products/view/sdm630modbus

[10] A value close to 1 indicates that the variables tend to increase together, while a value close to 0 indicates that they are independent.

the complexity of the models to be developed. Quality of prediction w.r.t. the sensed power consumption is measured for all models in terms of *mean squared error (MSE)*, defined as: $MSE = \frac{1}{n}\sum_{t=1}^{n}(y_t - \hat{y}_t)^2$, where y is the actual power consumption received from the sensors at time t, $\hat{y}$ is the predicted power consumption, and n is the number of samples under analysis.

Power model of the quality check station

The QC station is always active, as the only monitored devices are the workstation and the camera, which is always on. As a result, its power consumption is quite stable, and it is normally distributed (mean 110.69W) with a small standard deviation ($\sigma = 1.29W$).

To predict its power consumption, we thus adopted a mathematical model that allows to identify any behavior not included in the range of the expected ones. The model relies on a *moving average algorithm*, where the predicted value $\hat{y}$ is the mean of the previous n data points. The choice of the size n of the sliding window depends on the desired smoothing: an increased value of n enhances the smoothing at the cost of accuracy. The estimated power consumption at time t is therefore computed as:

$$\hat{y}[t] = \frac{y[t-1] + y[t-2] + ... + y[t-n]}{n} \tag{6.1}$$

where y is the measured power consumption in the previous time steps at time $t-1...t-n$, and $\hat{y}$ is the predicted power consumption. This model assumes that consecutive power demand samples have a similar consumption; as such, most of the estimate depends on the latest measurement with a defined sliding window ($n = 15$).

Historical data are then used to derive the typical *standard deviation* of the distribution σ. The resulting power model thus exploits both the moving average and the standard deviation: at any time instant t, the moving average is used to predict power consumption $\hat{y}[t]$: if the sensed power consumption falls in the range $\hat{y}[t] \pm 3 \cdot \sigma$ then the sensed value is considered coherent with the estimation, else an alarm is set. This mechanism is exemplified in Figure 6.17, where the red line is $\hat{y}$, the shaded area highlights the allowed prediction interval over time, and the blue line are the measured values y. The achieved MSE is very low ($1.23W$), thus achieving a very good prediction accuracy.

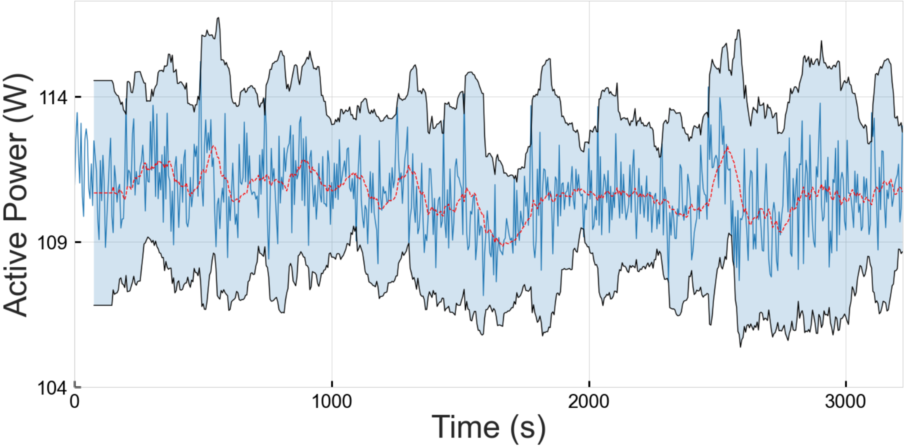


Fig. 6.17: Power model for the quality check station: active power (blue line), estimated power consumption $\hat{y}[t]$ (Equation 6.1), and allowed prediction interval $\hat{y}[t] \pm 3 \cdot \sigma$ (light blue area).

Power models of the robot assembly station

The ABB Yumi and the Kuka LR robot arms are used to pick and place LEGO-like blocks from the pallet, so to assemble them. Their task is thus periodic: when the pallet enters the bay, each arm picks one block from the pallet, puts it in the correct position, and then goes back to idle. The resulting power consumption curves have peaks in correspondence to each pick or place phase (Figures 6.18 and 6.19).

For both robots, the data infrastructure exports the status and the operating mode of the robot (idle, active and pick, active and place), plus the measured power consumption. The available historical data is used to construct a model of power consumption based on a *neural network* that creates a relationship between input parameters (i.e., the $Idle$, $Pick$, and $Place$ operating modes of the robot arms) and the output O, i.e., the estimated power consumption. The neural network has two hidden layers with 32 and 64 neurons, respectively, and one output layer that predicts the power consumption with 300 epochs used to update the weights and the bias of the parameters by optimizing the error between the training sample with the predicted power. For each robot, the model has been trained using 80% of the available historical data by using the Levenberg–Marquardt back-propagation training algorithm (*training set*) and verified using the remaining 20% (*test set*). The estimated MSE is $13.21W$ for the Kuka LR and $2.11W$ for the ABB Yumi: the model thus proves to the accurate in the estimation of the power consumption of the robots. Figures 6.18 and 6.19 show a snapshot of the evolution of the models by reporting measured vs. predicted power consumption (top, solid, and dashed, respectively). The power model generates an alarm if the measured power consumption does not fall in the range $\hat{y} \pm MSE$, thus allowing a tolerance threshold to take into account the possible prediction error.

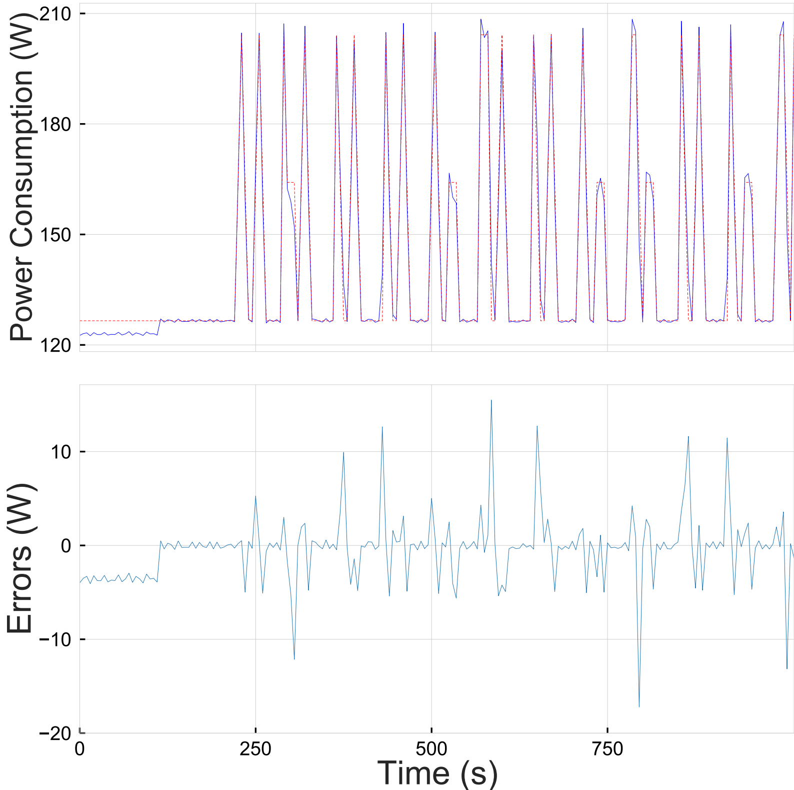


Fig. 6.18: Kuka measured (solid blue) and estimated (dashed red) power consumption (top) and error of the prediction model (bottom).

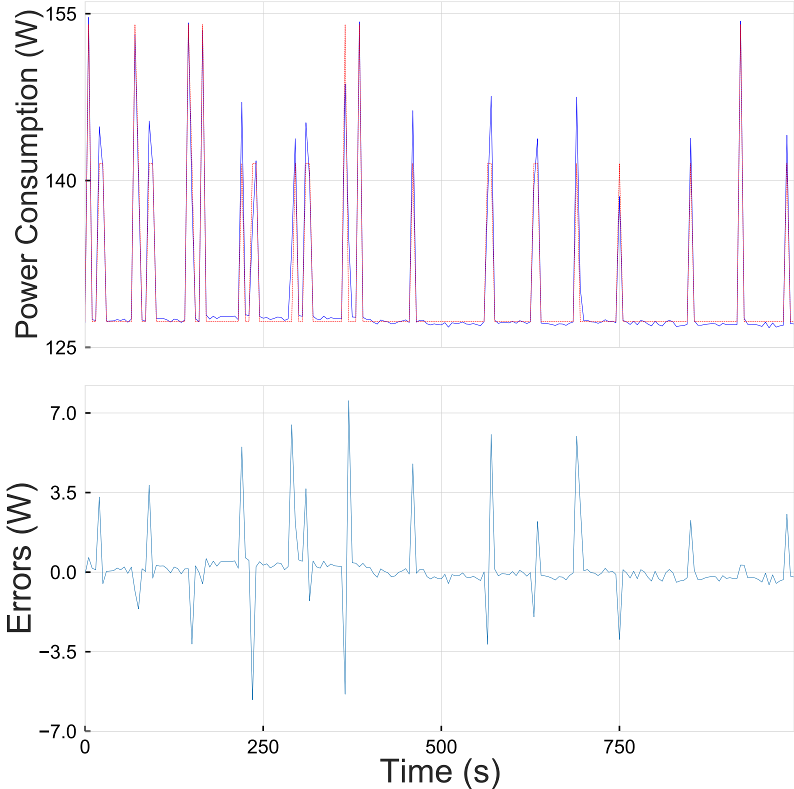


Fig. 6.19: ABB measured (solid blue) and estimated (dashed red) power consumption (top) and error of the prediction model (bottom).

Power model of the transport line

The transport line movement is managed by 15 motors (Figure 6.20), of which two control the main belts, while the others allow movement from the main belts to the bays. Sensor data monitors the overall power consumption of the transport line, thus aggregating not only the three AC phases but also the power consumption demand of all motors. To derive an accurate model of power consumption, it thus becomes crucial to know which motors are active at any time. For this experiment, we activated only the belts of interest for the monitored devices, i.e., motors $M1$ and $M2$ that control the main belts, plus motors $M12$, $M7$, and $M8$ that move the belt connecting the bay with the robot assembly station. Reducing the number of motors active at any time allows indeed to reduce the noise and control the power consumption better, given that we get an aggregated power consumption curve for all motors. The data received from the MES allows for knowing what motor is active at any time (green labels). We thus analyzed historical data on power consumption jointly with information about active motors overtime to separate the contribution of each motor to the overall power consumption. Then, we built a *moving average model* similar to the one in Section 6.3.4, that takes into account also which motors are active at any instant. Motors $M1$ and $M2$ are always active; thus, the base predicted moving average is $\hat{y} = \mu_{M1} + \mu_{M2}$. When one motor Mi becomes active, the current overall moving average is increased by the mean power contribution of the motor μ_{Mi}, so to build the power prediction by also considering its contribution to power consumption; vice versa, when a motor Mi becomes idle, the mean of its contribution to power consumption μ_{Mi} is subtracted from the current overall moving average, to make a prediction that considers this motor as turned off.

To exemplify this process, Figure 6.21 shows the evolution of sensed power consumption (blue line), moving average $\hat{y}$ (red), and allowed prediction interval when turning on and offline motors alternatively

(green labels). At the beginning, only motors $M1$ and $M2$ are active, thus $\hat{y} = \mu_{M1} + \mu_{M2}$. Then, motors $M7$ and $M8$ are activated to move the pallet to the robot assembly bay: $\hat{y}$ is increased with the estimation μ_{M7}, and then of μ_{M8}. When the motors are turned off, the estimation goes back to $\hat{y} = \mu_{M1} + \mu_{M2}$. Then, the prediction follows the subsequent activation of the motors. This model allowed to reach an MSE of $5.58W$, thus achieving a very good prediction accuracy.

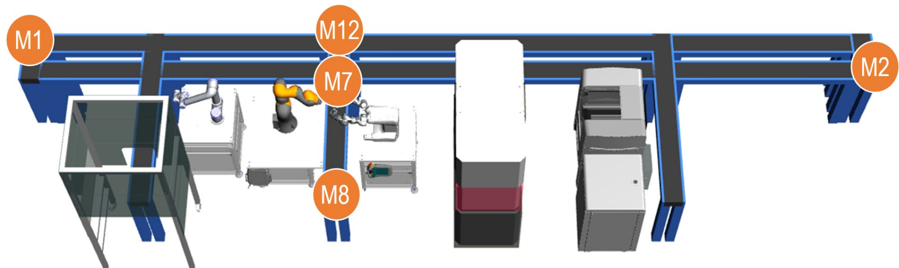


Fig. 6.20: Organization of the transport line. The numbers identify the motors monitored in this analysis: the main belt motors ($M1$ and $M2$) and the motors going to the robot assembly bay ($M7$, $M8$, and $M12$).

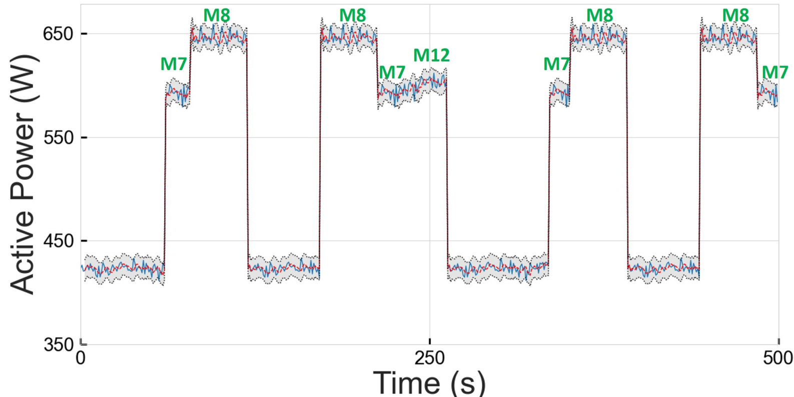


Fig. 6.21: Transport line Power Consumption (blue line) and application of the prediction model: the red line is the estimated power consumption $\hat{y}[t]$, while the light blue area represents the allowed prediction interval. Motors $M1$ and $M2$ are always active; green labels indicate the activation of additional motors.

Adoption of the digital twin for anomaly detection

The built models for power consumption concur in building a digital twin of the production line. The digital twin runs in parallel with the line operation: it is fed with commands and sensed data, it estimates the expected power consumption of the single devices based on the developed models, and it compares the prediction $\hat{y}$ w.r.t. the sensed data. The data processing infrastructure can be used to build services, like graphical rendering of the real-time power consumption and of the predicted consumption. Additionally, the digital twin allows live detection of unexpected behaviors, identified as a misalignment between the sensed and the predicted power consumption: in this case, an alert is notified to the user that can monitor the operation of the device of interest to identify any possible wear and tear effect, or to discard the alert as an anomalous sensed data.

Figure 6.22 shows an example of this for the Kuka robot arm. This figure is similar to Figure 6.18; in fact, it reports the real-time power consumption of the robot arm (solid blue line) and the corresponding

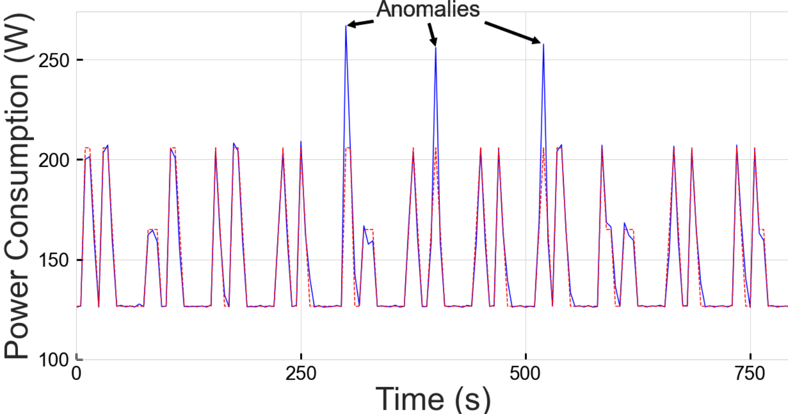


Fig. 6.22: Example of adoption of the digital twin of power consumption for anomaly detection: the unexpected power peaks of the Kuka robot are detected and used to fire an alert to the user and to the production line.

prediction of the model presented in Section 6.3.4 (dashed red line). In this case, the model detects three anomalous behaviors of the robot, i.e., three peaks that consume up to 30% more than the predicted value: these anomalies are detected by the power model, which sends an alert both to the plant and to the user interface. This allows the user to analyze the robot's behavior and identify any disturbance in data monitoring infrastructure or any malfunctioning of the robot arm, thus allowing the application of monitoring and maintenance actions to prevent further damage to the equipment.

6.4 The design of a digital twin for predictive maintenance

Nowadays, terms like a smart factory, digital twin, and industry 4.0 [289] announce the dawn of a new era for intelligent automation. Computers should be able to monitor a real plant and make decisions to optimize production and reduce costs [290]. ICPSs are one of the most important key enabling technology used to represent each node of the production line [291]. This requires analyzing data from the real plant and to integrate them into a model representing the factory, usually called a digital twin. The concept of a digital twin is not new, but in these years, it has become strictly correlated to manufacturing processes [289]. The Digital Twin has the global view of the entire production line allowing data analysis and strategies planning targeted to maximize the production with real constraints coming from the real factory [290]. However, actually, it does not exist a reference development model that leads to the creation of a digital twin. One of the most critical points of this problem regards the health status analysis of equipment and reflects this information in the related model. The encapsulation of this information inside the model of a digital twin is usually a hard challenge that requires a lot of time spent to refine the entire model. For instance, this information could be retrieved from the technical knowledge of vendors, equipment maintainers, or data analysis coming from sensors. All the mentioned strategies are not easily and quickly adaptable to the equipment model.

In literature, many works tried to solve this issue by proposing ad-hoc solutions for specific cases. However, these solutions do not define a standard way of working to face the problem. In [5], some concepts have been discussed to engage the problem with a higher level of abstraction. The idea proposed is the reuse of techniques coming from the Electronic Design Automation (EDA) domain adapted to the automation domain. The EDA way of working tried to define a unique path that starts by formalizing the problem, moves to the solution by analyzing design automation techniques, and finally defines new standards. This approach is applied with a particular focus on the health status analysis of equipment that can be reused

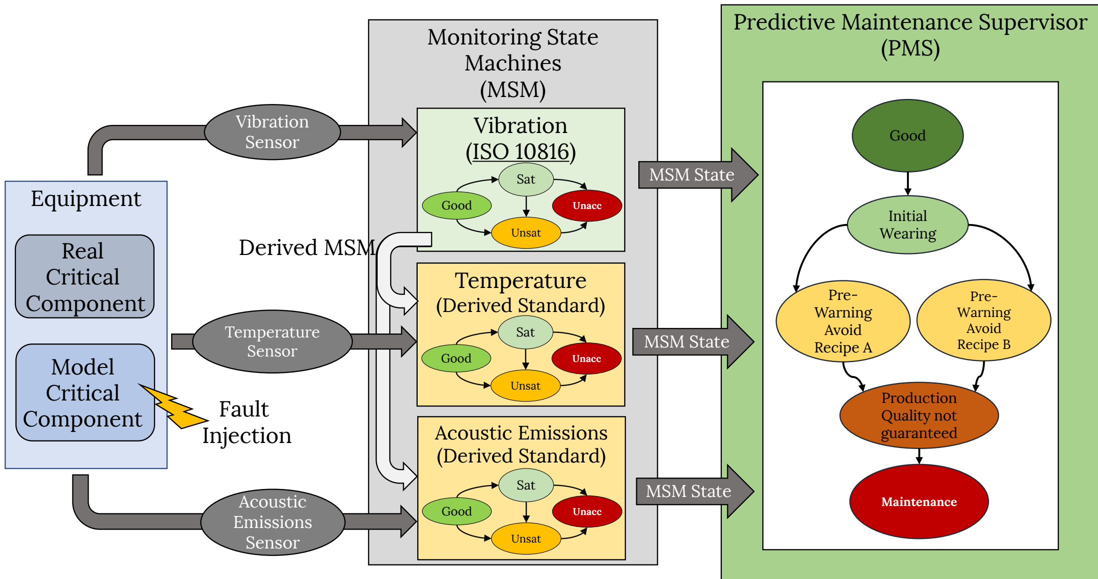


Fig. 6.23: Overview of the proposed approach.

in different scenarios. In particular, a novel approach that leads to the definition of equipment health status based on observable measurements and finite-state machines is presented. This approach describes a new architecture that enables the monitoring of industrial assets able to support the predictive maintenance task when multi-sensors are used to monitor equipment.

Figure 6.23 summarizes the proposal of creating a digital twin to support predictive maintenance for a ICPS. The entire flow starts from the data generation that can come from sensors of a critical component in real equipment or a model (vibrations, temperature, and acoustic emissions). First, the vibration data are collected and integrated into a finite state machine called Monitoring State Machine (MSM) that defines the severity level, based on ISO-10816, of the actual measurement. Then, the comparison between the vibration MSM and data coming from other sensors (temperature, acoustic emissions) allows defining MSM for other observable measurements. Finally, Predictive Maintenance Supervisor (PMS) is the actor that monitors all the MSMs and defines the health status of the critical component of the equipment under monitoring. The use of a model is fundamental to refine the PMS, but more importantly, it can be used to perform predictive maintenance via simulation and faults injection on the model. Moreover, the identification of the health status of the equipment allows us to reflect this information in the model and obtain the digital twin. The main contributions of the proposed solution are:

- Definition of a design flow for predictive maintenance-oriented digital twin;
- Flexibility to be adapted to other equipment;
- Reuse of existing standard (ISO-10816) to define new standards for other observable measurements;
- Evolution from condition-based to predictive maintenance via simulation with a unique framework.

6.4.1 State of the art and definitions

This section gives an overview of OPC UA [30] and FMI [169] that are the main technologies used in this work. Moreover, a comparison with state-of-the-art solutions is presented.

OPC UA standard

OPC UA represents a de-facto standard in industrial automation; it has been developed by the OPC Foundation. The goal of the standard is to unify the communications between different actors of a manufacturing plant. These actors can cover different positions in the automation pyramid. Figure 6.7 shows the automation pyramid with respect to the ISA-95/IEC 62264 standard [292]. The lowest level of the pyramid is level 0, where equipment performs physical operations. Level 3 and level 4 represent the two highest levels of the pyramid where respectively, ERP and MES schedule operations and perform data analysis of the production quality. In this context, the OPC UA standard enables communications between the different levels of the automation pyramid, both horizontally (machine to machine) and vertically (Machine to ERP and vice versa). The standard provides two different communication mechanisms: client-server and publish-subscribe. The standard defines a set of functions that allows exchanging data exposed from a server or publisher. Moreover, the standard defines a set of profiles. An OPC UA server that implements a certain profile allows using a set of specific functions defined by the specific profile. For instance, the `historical data access` profile allows retrieving historical data that is stored in the OPC UA server. Exposed data of each server are visible in a special structure called `information model`. OPC UA allows structuring the exposed data with the use of an object-oriented paradigm through an XML schema defined by the standard.

FMI standard

FMI is a standard born from MODELISAR cooperative in 2008, currently in its third revision. [11]The goal of the standard is to define a clear and usable interface for dynamic models. Usually, each modeling environment has its own interface that can be used to couple different tools or import external libraries. This requires adapting the native model depending on the target environment. Generally, this is a classic strategy used to couple different modeling environments. Unfortunately, the computational effort required to synchronize different tools is usually high. This promising standard allows exchanging models described in different languages enabling the cooperation of heterogeneous models. A FMU represents the basic block of the standard. The standard is based on the separation of interface and functionality. The FMI defines a set of C-APIs that wraps the model allowing it to interact with external models. More in detail, these functions define how to set or get a value from the interface of the model and how to simulate it. Then, the wrapped model is compiled as a shared library. The interface of the model is described through an XML file called `modelDescription.xml` that contains a well-defined XML schema. A FMU is a zip file that contains the obtained shared library and the *modelDescription.xml*. When a target environment imports a FMU, it can easily identify its interface from the XML file and interact with it through the C-APIs.

Related Works

The identification and prediction of faults of equipment have been studied in a multitude of researches [293, 294, 295]. Two principal classes of maintenance have been identified: time-based and condition monitoring. The main difference between these two classes depends on the maintenance strategy. The first relies on periodical maintenance cycles, while condition monitoring relies on the status of the equipment. Concerning time-based maintenance, failure models have been defined to estimate the failure percentage of equipment. However, these failure models are not accurate enough. In fact, the precision of these models detects only 11% of all the failures [296]. In [297], the authors analyze the correlation between vibration and wear debris that can occur from mechanical wear. Yaqub et al. [298] proposed an interesting approach based on

[11] https://fmi-standard.org/

Machine			Class I Small Machines	Class II Medium Machines	Class III Large Rigid Foundation	Class IV Large Soft Foundation
Vibration Velocity Vrms	in/s	mm/s				
	0.01	0.28				
	0.02	0.45				
	0.03	0.71		GOOD		
	0.04	1.12				
	0.07	1.80				
	0.11	2.80		SATISFACTORY		
	0.18	4.50				
	0.28	7.10		UNSATISFACTORY		
	0.44	11.20				
	0.70	18.00				
	1.10	28.00		UNACCEPTABLE		
	1.77	45.90				

Fig. 6.24: ISO 10816: Definition of Vibration Severity Levels with respect to the machine class

the classification of vibrations into severity levels. This is a promising approach, but it is strictly related to vibration analysis. The combination of different observable measurements has been studied in [299]. The section explains how the combination of vibration and temperature of a mechanical system allows for better detect the health state of the system. In [300], a tracking simulator is presented, which relies on the estimation of the dynamic process through data coming from the OPC UA server of the physical process. This last work shows an approach to refine a virtual model of the equipment but does not consider the correlation of different observable measurements (vibrations, temperature, *etc*).

6.4.2 From sensor data to severity levels

The observation of mechanical equipment is the mainly technique used to identify its health status. This can be achieved through the observation of measurements coming from sensors. Different research [301] have been focused on the development process of IoT sensors (accelerometers, gyroscopes, pressure sensors, etc.). Some works tried to define thresholds that can be used as guidelines to identify the severity of the measurement deviation. For example, the International Organization of Standardization (ISO) has proposed a standardization for the vibration measurements of mechanical parts. This standard tries to define a set of severity levels combining vibrations with the class of machines [302]. The standard establishes general procedures for the measurement and evaluation of mechanical vibration of machines, as measured on non-rotating parts(i.e. casing of a motor).

$$v_{r.m.s.} = \sqrt{\frac{1}{T}\int_0^T v^2(t)dt} \tag{6.2}$$

The $V_{r.m.s.}$ (vibration velocity root mean square) is obtained with the equation 6.2. T is the sampling time, and it is correlated with the vibration sensor frequency. v is the time-dependent vibration velocity, and V_{rms} is the corresponding r.m.s. velocity.

Figure 6.24 reports the classification of the ISO-10816. It considers four class machines depending mainly on the generated power.

- *Class I*: Small machines that generates up to 15kW output;
- *Class II*: Medium-sized machines, without special foundations, with an output up to 75kW;
- *Class III*: Large prime movers with rotating masses mounted on rigid and heavy foundations;
- *Class IV*: Large prime-movers and other machines with rotating masses with an output greater than 10MW (i.e. turbogenerator, gas turbine).

The standard defines four different states: *good*, *satisfactory*, *unsatisfactory*, and *unacceptable*.

- *Good*: The vibration of newly commissioned machines normally falls within this zone;
- *Satisfactory*: Machines with vibration within this zone are normally considered acceptable for unrestricted longterm operation;
- *Unsatisfactory*: Machines with vibration within this zone are normally considered unsatisfactory for long-term continuous operation. Generally, the machine may be operated for a limited period in this condition until a suitable opportunity arises for remedial action;
- *Unacceptable*: Vibration values within this zone are normally considered to be of sufficient severity to cause damage to the machine.

In particular, it is important to notice that with the same $V_{r.m.s.}$, the different machine classes could have different states. For instance, let consider the $V_{r.m.s.}$ range between 7.10 and 11.20 mm/s. A large machine of *class IV* with this range is in a state of *satisfactory* and a small machine of *class I* is in an *unacceptable* state.

The mentioned standard is applied only for vibration measurements. Actually, there is no other standard that defines severity levels for other observable measurements (i.e. temperature, acoustic emissions, power consumption).

This work shows a novel approach to defining severity levels for other observable measurements relying on the ISO-10816 existing standard. More in detail, the approach combines vibrations with other observable measurements and retrieves the related states mapped from ISO-10816. This can be applied if and only if other observable measurements are strictly related to a particular component of the considered machine. Table 6.25 shows the theoretical application of this novel approach with a machine of *class I*.

6.4.3 Monitoring State Machine (MSM)

The goal of predictive maintenance is to anticipate equipment failures to optimize production and plan maintenance strategies. In this scenario, an equipment maintainer can establish the status of equipment by observing it through measurements of vibrations, temperature, power consumption, etc.. When these measurements deviate from the nominal behavior, the equipment could perform inaccurate operations or stop for failure. With the analysis of these measurements, it is possible to define the state of health of the equipment.

An MSM is a state machine that monitors one observable measurement of a critical equipment component and translates raw data coming from sensors into severity levels representing its health status (see figure 6.26). This work relies on the concept of severity levels defined by ISO-10816 for vibration measurements and adapting it to other observable measurements. An MSM is composed of four states explained in section 6.4.2 (good, satisfactory, unsatisfactory, unacceptable) defined by the ISO-10816 standard. The use of ISO-10816 coupled to a finite state machine guarantees persistent deviation detection avoiding transient

Machine				Vibration Severity Level	Temperature Severity Level
	Vibration	Temperature	Acoustic Emission		
Measurements	0.28	...	...		
	0.45	...	...		
	0.71	...	...	GOOD	
	1.12	...	...		
	1.80	...	...		
	2.80	...	...	SATISFACTORY	
	4.50	...	...		
	7.10	...	...	UNSATISFACTORY	
	11.20	...	...		
	18.00	...	...		
	28.00	...	...	UNACCEPTABLE	
	45.90	...	...		

Fig. 6.25: Definition of severity levels for other observable measurements. In this case, temperature and acoustic emissions are considered.

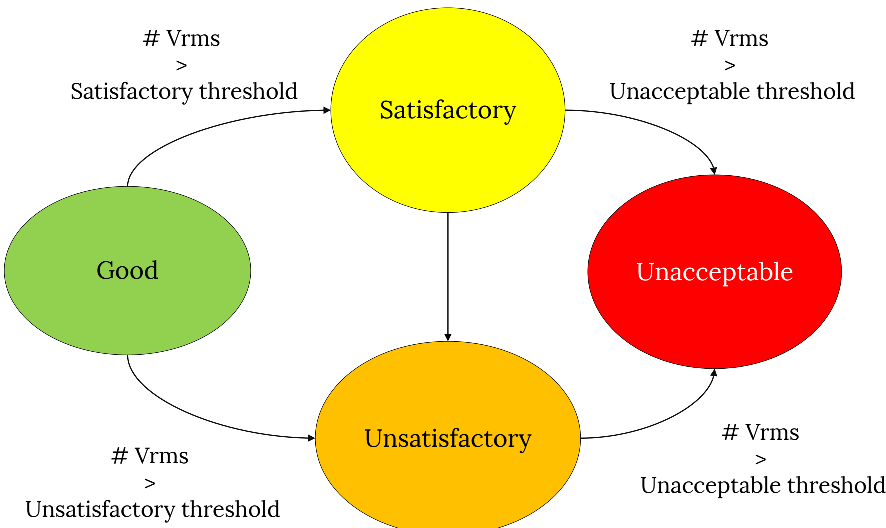


Fig. 6.26: Overview of a generic MSM. The states reflect the severity levels defined by ISO-10816.

events. Transient events could occur from sensing errors leading to wrong equipment diagnosis. A persistent deviation can be detected from an MSM with a transition between one state to another, triggered when a severity level occurs for a certain amount of time. The transition can be defined by a threshold that can be set during the tuning phase of the MSM. The tuning phase requires a model of the critical component of the equipment that will be faulted. This phase will be explained in section 6.4.5. Figure 6.26 represents a vibration MSM. Let us assume that the vibration MSM is in a good state. The vibration MSM will move to a satisfactory state only after a persistent detection of a satisfactory severity level, defined by the satisfactory threshold. Unfortunately, there are no similar standards of ISO 10816 for other observable measurements. The definition of MSMs for other observable measurements of the same critical component is performed via the correlation of vibration severity levels with other observable measurements (see Figure 6.25). For instance, in Figure 6.25, the correlation of vibration severity levels, temperature, and acoustic emissions of

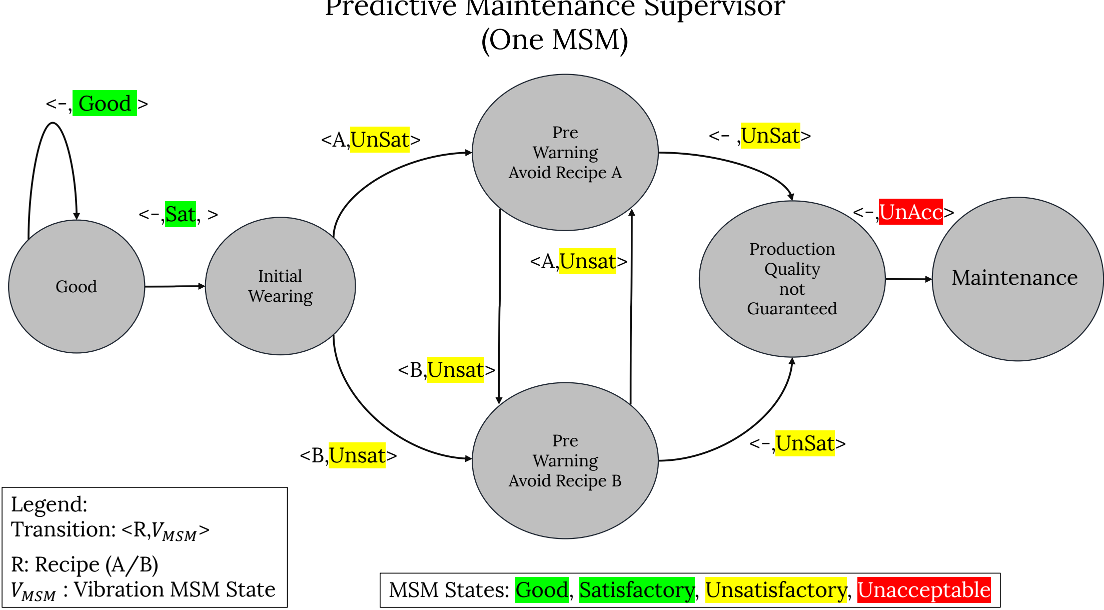


Fig. 6.27: Overview of a Predictive maintenance supervisor monitoring one MSM (vibration MSM).

the same critical component allows having more information about its health status and defines the state range for each measurement.

6.4.4 Predictive Maintenance Supervisor (PMS)

In a real factory, a maintainer tries to classify the health status of equipment by observing its measurements. For instance, the maintainer could identify machine states like "equipment working", "equipment deteriorating", "maintenance required," and "broken". This is usually performed manually with the personal knowledge of the maintainer acquired with the experience. In the previous section, MSM has been presented with the role of monitoring the deviations of observable measurements of the same critical component. This section introduces a new actor called Predictive Maintenance Supervisor (PMS). The role of this actor is to monitor all the MSMs to detect anomalies and to raise alerts regarding health status.

The PMS is defined as a finite state machine that analyses the current state of the available MSMs. It is modeled as a transition system where the output of the automaton is its current state.

The inputs of the PMS are represented with the tuple $< R, M >$. *R* represents the class of recipes that are produced by the equipment. For instance, the equipment produces two different products with two different recipes A, B that are different in terms of energy consumption and mechanical stress of the critical component. This input is necessary to model better the PMS, allowing to catch more information about the real health status of the equipment. *M* represents the class of the available MSMs monitored by the PMS. In particular, the PMS takes as inputs the internal state of the MSMs (good, satisfactory, unsatisfactory, unacceptable). Figures 6.27 and 6.28 show two different structures of PMS. Figure 6.27 shows the structure of a basic PMS that monitors only one MSM, which detects vibrations severity levels. In this basic PMS, it is possible to notice the following states: *good, initial wearing, avoid recipe A, avoid recipe B, production quality not guaranteed, maintenance* that reflects the health status of the critical component. The execution of the PMS starts from *good* state. The PMS will stay in a good state as long vibration MSM is in *good* state regardless of the recipes (transition $< -, Good >$). When the vibration MSM detects a persistent deviation coming from the vibration sensor, it will move to *satisfactory* state. This implies a transition

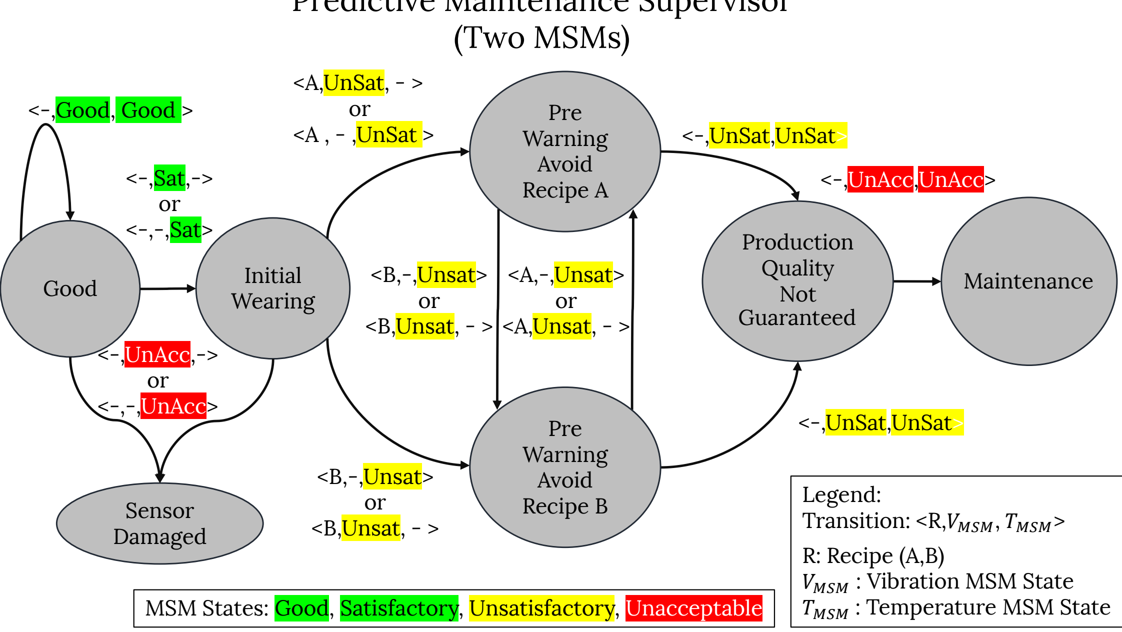


Fig. 6.28: Overview of a Predictive Maintenance Supervisor (PMS) monitoring two MSMs (vibration and temperature).

for the PMS from *good* to *initial wearing* state (transition $< -, Sat >$). When the critical component has a significant deviation, the MSM moves to an unsatisfactory state. From the PMS point of view, this means that the critical component is generating significant vibrations with the actual recipe. In this case, the PMS moves into *avoid recipe A or B* state, depending on the actual recipe. This is due to the fact that the two recipes are different in terms of mechanical fatigue, and this is reflected in the critical component in terms of vibration severity. For instance, let us assume that recipe A generates more mechanical stress than recipe B. If the equipment generates more vibration with recipe A the PMS notifies that recipe A should be avoided (transition $< A, Unsat >$). This can improve the use of the equipment instead of going directly to maintenance, reducing undesired equipment downtime. When the PMS detects an unsatisfactory severity level from the vibration MSM, for both recipes A and B, it moves to *production quality not guaranteed* state (transitions $< A, Unsat >, < B, Unsat >$). This means that the critical component has shown a continuous and important deviation in all the possible recipes. The PMS can not detect the quality of the production because it does not have enough information related to the output products from the equipment. In this case, the PMS does not move directly to *maintenance* state in order to avoid equipment downtime. From a production planning perspective, this is important information that could help to define different quality levels of the products. For instance, products that are produced when the equipment is *production quality not guaranteed* state could still be good but require additional visual or technical analysis. The *maintenance* state is reached from the previous state when the PMS detects unacceptable severity levels from MSM.

Figure 6.28 shows the structure of a PMS that is monitoring two MSMs, vibration, and temperature. In section 6.4.3, the definition of severity levels starting from ISO-10816 for other MSMs have been discussed. The structure of this PMS with a second MSM allows the PMS to retrieve additional information than the previous PMS (figure 6.27. The two MSMs are considered part of class M in the tuple $< R, M >$ used to represent the PMS state transitions. In particular, a new state called *sensor damaged* has been added that represents sensor anomalies that can be detected from the PMS. Let us consider the PMS in *good* or *initial wearing* state and an anomaly in the vibration sensor. The vibration sensor is reporting an important and

persistent untrue deviation that is translated into an unacceptable severity level from the vibration MSM. The PMS detects this deviation from vibration MSM and a good severity level from temperature MSM. In this case, the PMS moves to *sensor damaged* state.

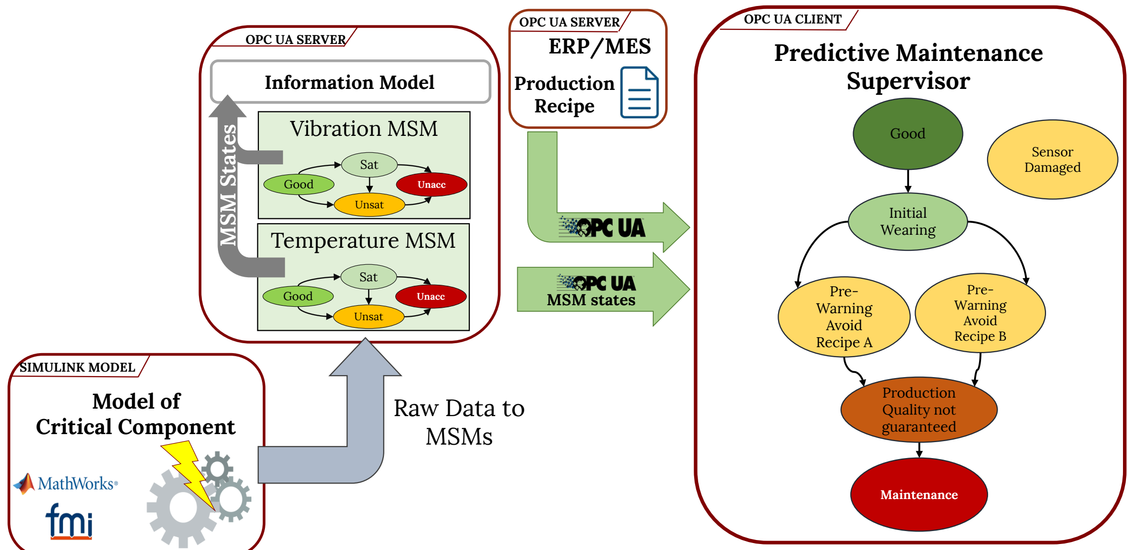


Fig. 6.29: Overview of the entire framework in validation mode.

6.4.5 Alternative use of MSM and PSM

This section discusses the alternative use of the framework composed of MSM and PMS in different scenarios. The entire framework has been wrapped in an OPC UA client and server in order to be easily integrated into a production plant environment and retrieve the necessary data in the different following scenarios (see Figure 6.29). In particular, all the MSMs were integrated into the OPC UA client and the PMS on the server side.

Structure Validation Mode

In this scenario, we assume that a model of the equipment's critical component is available. Clearly, the model is an abstraction of the real equipment, and it can not be precise as the real measurements coming from the equipment sensors. However, the model is used to define preliminary new standards for the observable measurements and then set the internal thresholds of all the MSMs (see section 6.4.3). Moreover, the *Severity Level Classifier* needs to log the normal behavior for each product produced from the equipment. The model is exported as an FMU and then integrated into an OPC UA server. The *Severity level Classifier* is integrated into the OPC UA Server and linked to the observable measure of the model. The information model of the OPC UA server exposes only the MSMs states that are needed for the PMS. For instance, if vibration and temperature are the observed measurements, the information model will contain only the vibration and temperature MSM actual states. The OPC UA client, which contains the PMS, retrieves the severity levels from the OPC UA server information model. Moreover, the validation phase requires to fault the model in order to check if the framework detects the deviation of the faulted model.

Condition-based-Maintenance Continuous Mode

The framework is integrated into the real equipment after the validation phase. The OPC UA Server that contains the MSMs is connected to sensors that retrieve data from the real equipment. The data from the sensors are related to the actual product that is produced by the equipment. The framework in the OPC UA client monitors the real equipment and raises alerts when an anomaly is detected during production. In this scenario, the framework is used to perform Condition-based Maintenance. This is due to the fact that the framework receives data regarding the actual product. The framework can predict the health status of the equipment when the "symptoms" are just detected.

Predictive Maintenance Structure Training Mode

This mode represents a preparatory phase for the *Predictive Maintenance Mode*. This phase is performed concurrently with the *Condition-based-Maintenance Continuous Mode*. The framework monitors the equipment and stores raw data related to the severity levels and the type of product that is produced. This tuple is stored in a local or remote database.

Predictive Maintenance Mode

A factory can produce different types of products that can affect the health status of the equipment depending on the physical process required from the product recipe. Let us assume that a Factory produces three different products A, B, and C. For instance, product A requires less mechanical stress than product B or product C. The goal of the Framework in Predictive Maintenance Mode is to predict if a production batch can be produced without anomalies. The information on the production plans can be retrieved from ERP or MES that cover the highest levels of the Automation Pyramid.

The framework in the OPC UA client retrieves this information from the ERP/MES OPC UA server. The production plans contain the type of products (A, B, C) and the number of products to produce. The combination of production plans and the historical data stored in the *Predictive Maintenance Structure Training Mode* allows using the framework offline to perform predictions related to the specific production plan. The framework simulates the MSMs and the *Supervisor*, using the local raw data stored in the database.

6.4.6 Methodology application

This Section explains the methodology application and the experimental results.

Experimental Setup

The methodology has been validated with the use of two support technologies: OPC UA and FMI. The two principal components that compose the framework of the experiments are the OPC UA client and server (see figure 6.29). In particular, the two actors have been developed in two different FPGAs. Now the experiment setup necessary to validate the methodology is described:

OPC UA Client

The core of the methodology is implemented into the OPC UA client. The client has been developed in python with the use of FreeOpcUa python library[12] on a PYNQ-Z1 FPGA. This choice comes from the

[12] http://freeopcua.github.io/

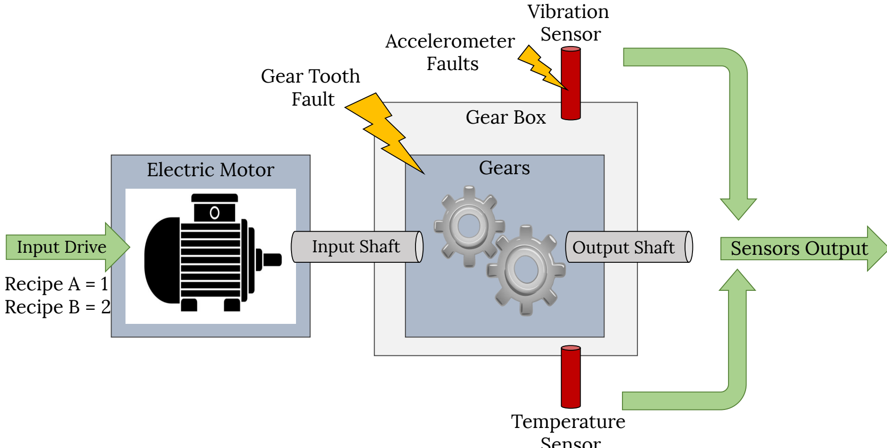
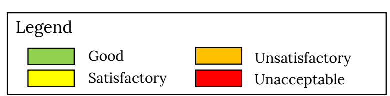


Fig. 6.30: Overview of the transmission system. The gears represent the critical component of the system. The model can be faulted in two different parts: the gear tooth and the vibration sensor.

necessity to have a specific description that can be synthesized to an FPGA to obtain a compact reactive device minimizing power consumption. The PMS has been described using a Hardware description language (Verilog) and then synthesized into the FPGA that provides python API to perform data exchange with synthesized bitstream[13].

a) Severity Levels Correlation

Tooth Fault Level (%)	Recipe A		Recipe B	
	Vibration Vrms (mm/s)	Temperature (°C)	Vibration Vrms (mm/s)	Temperature (°C)
0	0.714	19.85	1	20,50
5	0.946	20.65	1.66	21.45
10	1.157	22.15	1.877	23.55
20	1.58	26.35	2.3	29.45
30	2.018	32.25	2.743	36.65
40	2.416	40.85	3.151	46.95
50	2.845	51.05	3.586	58.65
60	3.301	62.95	4.048	71.75
70	3.791	76.55	4.543	86.16
80	4.326	91.45	5.081	101.65
90	4.952	108.15	5.721	118.55
95	6.736	133.05	8.483	159.55
98	10.57	200.25	11.59	237.85
100	12.67	268.05	13.24	289.75



b) Temperature Severity Levels

Tooth Fault Gain (%)	Temperature	Severity Level Range
	(°C)	
0	19.85 20,50	≤ 20,50
5	20,65 21.45	20,51-40,85
10	22.15 23.55	
20	26.35 29.45	
30	32.25 36.65	
40	40.85 46.95	40,86 - 133.05
50	51.05 58.65	
60	62.95 71.75	
70	76.55 86.16	
80	91.45 101.65	
90	108.15 118.55	
95	133.05 159.55	>133.05
98	200.25 237.85	
100	268.05 289.75	

Table 6.2: Definition of new standard for temperature MSM. Table I-A reports the experiments for recipes A and B with different tooth fault levels with respect to ISO 10816 severity levels. Table I-b shows the obtained severity level ranges for temperature MSM.

[13] http://http://www.pynq.io/

OPC UA Server

The server has been developed in a second FPGA using the same structure presented for the client. All the MSMs have been developed in Verilog and then synthesized on the FPGA. Then, the FREEopcUa python library has been used to develop the OPC UA server information model where all the MSM actual states are exposed. The OPC UA server retrieves data directly from real sensors attached to the critical component of the equipment or from a model of the critical component that is not integrated into the OPC UA server. The model is often complex, requiring a computational effort that can not be computable by a small device. The server is connected remotely to the model via a socket connection. The model is exported using the FMI standard as an FMU to be as flexible as possible. This allows having more control over the model that can be easily reallocated to other remote resources or integrated with other environments.

Abstract Model of a Real Machine

A driveline two-speed transmission system is used as a model of a critical component of an industrial equipment (see figure 6.30). Mathworks Simulink © is used to model the critical component.

The model consists of an electric motor, a gear system contained in a gearbox, and two shafts. The electric motor is connected to the gearbox via the input shaft. The gearbox contains two gears with different dimensions that are connected respectively to the input and output shaft. In particular, the gears in the gearbox represent the critical subsystem of the manufacturing equipment. The model incorporates two sensors to monitor vibration and temperature. The two sensors are attached to the surface of the gearbox, monitoring the vibration displacements and the external temperature.

The model can be faulted in two parts of the critical component. These faults are included in the faults presented in Chapter 4 because are parametric faults. These parametric faults are controlled variations of three different internal parameters of the model under test that are related to physical variations of the mechanical properties of the model. Thus, the fault taxonomies proposed in Chapter 4 are more fine-grained. The multi-domain fault approach manipulates or adds differential equations directly into the models to inject specific faults, one fault at a time. Consequently, the approach presented in this section is more general because it was developed earlier than all the other methodologies proposed in this thesis. The tooth of the gear and the vibration sensor can be faulted to simulate the wearing of the gears or sensors anomalies. The gear tooth fault is driven by an integer input with a range from 0 to 100 that represents the percentage of fault. The parameter affects the gear tooth, which will increase the gear vibrations. The vibration sensors can be faulted with two parameters: gain and offset. The gain parameter can be used to simulate an amplification or attenuation of the output sensors data, and offset allows to add an offset to these data. With these two parameters, it is possible to model the principal effects caused by sensor anomalies. The model is driven by a single real input that defines the output speed of the electric motor. This model represents a critical subsystem of manufacturing equipment. The manufacturing equipment produces two different products classified with recipes A and B that are represented by two different speeds of the electric motor, part of the critical component. Both recipes require 40 seconds to be executed by the critical component but generate different mechanical stress for the gears.

Structure Validation Mode in Action

This section describes the experiments used to set and validate the correctness of the methodology. In particular, the Structure validation mode of MSM is considered (see figure 6.29). Two different experiments have been considered:

- One MSM (vibration)
- Two MSMs (vibration and temperature)

For the first experiment, only the vibration sensor is considered. In this case, the setup phase is required only to set the MSM thresholds to detect the persistent deviation of vibrations. The severity levels used are those defined by ISO 10186 standard (see Figure 6.24). The tooth fault has been manually injected following the wearing curve of a real gear system. Figure 6.27 shows the PMS used in this experiment that was able to detect the deviation related to the gear tooth fault. After the setup phase, we tried to inject faults in the vibration sensor, and the PMS was not able to detect the sensor anomalies, as we expected. The PMS does not have enough information to clearly understand if the anomalies are coming from the sensor or the critical component.

In the second experiment, a second MSM related to the temperature has been introduced. In this case, the setup phase requires first to define a new standard for temperature MSM. Tables 6.2 a) and b) show the obtained severity level ranges for the temperature MSM. The PMS used in this experiment is the one presented in Figure 6.28). Once the new standard has been derived, and the thresholds for both the MSMs have been set, the PMS has been validated. The same wearing curve of a real gear system has been used to validate the methodology. The Supervisor has been capable of detecting gear tooth anomalies moving in different states. Then, the vibration sensor has been faulted, as in the previous experiment. In this case, the PMS was able to detect the sensor anomalies moving to *sensor damaged* state.

7

Conclusions and suggestions for future research

To conclude this thesis, a summary of the proposed methodologies built to support functional safety by generating faulty behaviors through different fault injection techniques is presented. Then, it suggests directions for future research that builds on top of the proposed approaches.

7.1 Summary of the proposed approaches

This thesis proposes state-of-the-art topics, addressing research gaps in terms of methodologies and tools for analysis, design, development automation, and safety evaluation of modern industrial systems. This complex task is accomplished by analyzing the behaviors of Industrial Cyber-Physical System (ICPS) in the presence of faults (objective `1` as defined in Section 1.2) in different physical domains, e.g., electrical and mechanical. The analysis of the faulty behaviors of an ICPS is subdivided in the thesis into four main research directions:

- analysis and modeling of analog faults into transistor-level descriptions;
- investigation and modeling of multi-domain faults into behavioral descriptions;
- abstraction of models described at different abstraction levels to be integrated into hierarchical digital twins;
- application to the industrial field by creating a data fusion infrastructure to correlate real data and models, i.e., by realizing a digital twin able to produce nominal and faulty behaviors.

The study proposed in this thesis started by analyzing the fault injection techniques used in a ICPS to assess the functional safety at the system level. Initially, fault models and injection techniques known as state-of-the-art for transistor-level descriptions are analyzed to understand the basis of the fault injection approaches. Based on the standard analog fault models described in the IEEE P2427 draft standard, two additional fault models are proposed to extend the standard, a stuck-on and -off models described at the transistor level. Moreover, automatic fault injection methodologies and fault grouping techniques are proposed to support the fault injection process and analysis of the results. All these methodologies are suitable for analog descriptions and are explained in detail in Chapter 3 and allow reaching the objective `1.A` of this thesis.

Starting from the knowledge acquired in the analog field, other physical domains are analyzed to understand how to model faulty behaviors. The Chapter 4 proposes different techniques to model and inject faults into electrical and mechanical descriptions modeled as differential equations. Furthermore, an automatic tool able to inject these faults into Verilog-AMS descriptions is proposed. These techniques suitable for multi-domain models allow reaching the objective `1.B`.

Then, Chapter 5 presents a methodology that allows abstracting the behavior of Verilog-AMS Piecewise Linear (PWL) descriptions to C++ and a second methodology suitable to abstract a generic behavior (both linear and nonlinear) of a transistor-level circuit. These abstraction methodologies are not limited to fault-free descriptions because if faults are added to the original descriptions, these faulty behaviors can be abstracted. These fault-free and faulty abstracted descriptions can be used to simulate heterogeneous Virtual Platform (VP), where different components described at different abstraction levels are combined together to simulate a complex system and improve the simulation speed by preserving the accuracy. Moreover, these methodologies are not limited to the electrical domain but could be applied to other physical domains, e.g., to mechanical descriptions to abstract its behavior. These abstraction methodologies allow reaching the objective `1.C`.

Finally, Chapter 6 present different methodologies able to correlate real data coming from a production line with synthetic data coming from the simulation. These models described at different abstraction level allows to model the digital twin of a ICPS at different levels based on the final usage of the twin. All these methodologies allow to reach the objective `1.D` and to conclude the thesis.

7.2 Directions for future research

Starting from the methodologies presented in this thesis, many other research directions can be followed. In detail, the possible future directions that can be followed to improve the state-of-the-art in the transistor-level and ICPS domain are:

- develop methodologies to automatically extract more detailed failure modes from faulty simulations waveforms to be included in an Failure Mode and Effect Analysis (FMEA) by exploiting the fault grouping techniques;
- develop methodologies to construct an FMEA directly from an analog schematic. For example, by exploiting the fault grouping techniques, it should be possible to automatically identify failure modes at the block level for analog blocks. Then one still needs to propagate them at the system level and check the outcomes versus safety goals.
- applies the multi-domain fault injection presented in Chapter 4 on other systems modeling languages, e.g., the Modelica language to analyze more complex multi-domain models already available from the Modelica community.
- adapts the abstraction methodologies presented in Chapter 5 for non-linear systems to mechanical systems for abstracting its behaviors.
- study how the diagnostic coverage calculated from a microcontroller that controls a physical system can vary in the presence of multi-domain faults injected in the physical model.

Summary of the proposed innovative contributions

This chapter reports the innovative contributions to the *state of the art* by the works proposed in this thesis organized in chronological order. The articles are grouped into four main topics as follows:

- the first group is related to *analog fault modeling and simulation at transistor-level*, which comprises the Chapter 3;
- the second group contains the articles related to *multi-domain fault modeling and simulation at the behavioral level*, which comprises Chapter 4;
- the third group is related to *abstraction from transistor-level to behavioral models* presented in Chapter 5.
- while the last group contains the articles related to *digital twin modeling and analysis*, which comprises the Chapter 6.

Analog fault modeling and simulation at transistor-level

1. **N. Dall'Ora**, S. Azam, E. Fraccaroli, A. Alberts, and F. Fummi, *"Predictive Fault Grouping based on Faulty AC Matrices,"* in 2021 24th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS). IEEE, 2021, pp. 11–16.
2. **N. Dall'Ora**, S. Azam, E. Fraccaroli, A. Alberts, and F. Fummi, *"A Common Manipulation Framework for Transistor-Level Languages,"* in 2021 Forum on specification & Design Languages (FDL). IEEE, 2021, pp. 01–07.
3. S. Azam, **N. Dall'Ora**, E. Fraccaroli, A. Alberts, R. Gillon, and F. Fummi, *"Investigation on Realistic Stuck-on/off Defects to Complement IEEE P2427 Draft Standard,"* in 2022 23rd International Symposium on Quality Electronic Design (ISQED). IEEE, 2022, pp. 51–57.
4. S. Azam, **N. Dall'Ora**, E. Fraccaroli, R. Gillon, and F. Fummi, *"Analog Defect Injection and Fault Simulation Techniques: A Systematic Literature Review,"* under review in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD/ICAS). IEEE, 2022, pp. 1–14.
5. **N. Dall'Ora**, S. Azam, E. Fraccaroli, R. Gillon, and F. Fummi, *"Verilog-A Implementation of Generic Defect Templates for Analog Fault Injection"* in 2023 33rd Great Lakes Symposium on Very Large Scale Integration (GLSVLSI). ACM, 2023, pp. 1–5.
6. **N. Dall'Ora**, S. Azam, E. Fraccaroli, R. Gillon, and F. Fummi, *"A Language Independent Framework to Manipulate Transistor-level Netlists,"* submitted to IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD/ICAS). IEEE, 2023, pp. 1–14.

Multi-domain fault modeling at behavioral level

7. **N. Dall'Ora**, S. Vinco, and F. Fummi, *"Functionality and Fault Modeling of a DC Motor with Verilog-AMS,"* in 2020 IEEE 18th International Conference on Industrial Informatics (INDIN), vol. 1. IEEE, 2020, pp. 35–40.
8. **N. Dall'Ora**, E. Fraccaroli, S. Vinco, and F. Fummi, *"Multi-discipline Fault Modeling with Verilog-AMS,"* in 2021 4th IEEE International Conference on Industrial Cyber-Physical Systems (ICPS). IEEE, 2021, pp. 237–243.
9. **N. Dall'Ora**, F. Tosoni, E. Fraccaroli, and F. Fummi, *"Inferring Mechanical Fault Models from the Electrical Domain,"* in 2022 IEEE 5th International Conference on Industrial Cyber-Physical Systems (ICPS). IEEE, 2022, pp. 01–08.
10. F. Tosoni, **N. Dall'Ora**, E. Fraccaroli, and F. Fummi, *"A Framework for Modeling and Concurrently Simulating Mechanical and Electrical Faults in Verilog-AMS,"* in 2022 Forum on Specification & Design Languages (FDL). IEEE, 2022, pp. 1–8.
11. F. Tosoni, **N. Dall'Ora**, E. Fraccaroli, S. Vinco, and F. Fummi, *"Multi-domain Fault Models for Industrial Cyber-Physical Systems,"* submitted to IEEE Transactions on Computers (TC-CS). IEEE, 2023, pp. 1–8.
12. F. Tosoni, **N. Dall'Ora**, E. Fraccaroli, S. Vinco, and F. Fummi, *"Thermal Digital Twin of a Multi-Domain System for Discovering Mechanical Faulty Behaviors,"* submitted to IEEE 21st International Conference on Industrial Informatics (INDIN). IEEE, 2023, pp. 1–8.

Abstraction from transistor-level to behavioral models

13. S. Azam, **N. Dall'Ora**, E. Fraccaroli, and F. Fummi, *"Functional Level Abstraction and Simulation of Verilog-AMS Piecewise Linear Models,"* in 2022 23rd International Symposium on Quality Electronic Design (ISQED). IEEE, 2022, pp. 39–44.
14. **N. Dall'Ora**, S. Azam, E. Fraccaroli, R. Gillon, and F. Fummi, *"A Complete Framework Moving from Transistor-Level to Behavioral Models,"* submitted to IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD/ICAS). IEEE, 2023, pp. 1–14

Digital twin modeling and analysis

15. S. Centomo, **N. Dall'Ora**, and F. Fummi, *"The Design of a Digital-Twin for Predictive Maintenance,"* in 2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), vol. 1. IEEE, 2020, pp. 1781–1788.
16. K. Alamin, S. Vinco, M. Poncino, **N. Dall'Ora**, E. Fraccaroli, and D. Quaglia, *"Digital Twin Extension with Extra-Functional Properties,"* in 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 2021, pp. 434–439.
17. **N. Dall'Ora**, K. Alamin, E. Fraccaroli, M. Poncino, D. Quaglia, and S. Vinco, *"Digital Transformation of a Production Line: Network Design, Online Data Collection and Energy Monitoring,"* IEEE Transactions on Emerging Topics in Computing, vol. 10, no. 1, pp. 46–59, 2021.
18. F. Tosoni, **N. Dall'Ora**, E. Fraccaroli, and F. Fummi, *"The Challenges of Coupling Digital-Twins with Multiple Classes of Faults,"* in 2022 IEEE 23rd Latin-American Test Symposium (LATS). IEEE, 2022, pp. 01–06.

19. S. Gaiardelli, **N. Dall'Ora**, F. Ponzio, E. Fraccaroli, S. Vinco, S. Di Cataldo, and F. Fummi, *"A Data Fusion Service-Oriented Infrastructure Enabling Automatic Anomaly Detection,"* submitted to IEEE Transactions on Industrial Informatics (TII). IEEE, 2023, pp. 1–8.

References

1. B. Kruseman, B. Tasic, C. Hora, J. Dohmen, H. Hashempour, M. van Beurden, and Y. Xing, "Defect Oriented Testing for analog/mixed-signal devices," in *2011 IEEE International Test Conference*. IEEE, sep 2011.
2. R. Drath and A. Horch, "Industrie 4.0: Hit or hype?[industry forum]," *IEEE industrial electronics magazine*, vol. 8, no. 2, pp. 56–58, 2014.
3. S. Gaiardelli, S. Spellini, M. Lora, and F. Fummi, "Modeling in industry 5.0: What is there and what is missing: Special session 1: Languages for industry 5.0," in *2021 Forum on specification & Design Languages (FDL)*. IEEE, 2021, pp. 01–08.
4. E. Sisinni, A. Saifullah, S. Han, U. Jennehag, and M. Gidlund, "Industrial internet of things: Challenges, opportunities, and directions," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 11, pp. 4724–4734, 2018.
5. N. Dall'Ora, S. Centomo, and F. Fummi, "Industrial-iot data analysis exploiting electronic design automation techniques," in *2019 IEEE 8th International Workshop on Advances in Sensors and Interfaces (IWASI)*. IEEE, 2019, pp. 103–109.
6. E. Lee and S. Seshia, *Introduction to Embedded Systems, Second Edition: A Cyber-Physical Systems Approach*, ser. Introduction to Embedded Systems. MIT Press, 2016. [Online]. Available: https://books.google.it/books?id=chPiDQAAQBAJ
7. A. Hopkins, "The functional safety imperative in automotive design," in *White paper*. ARM, 2016, pp. 1–7. [Online]. Available: https://www.design-reuse.com/articles/41670/the-functional-safety-imperative-in-automotive-design.html
8. A. Nardi, S. Camdzic, A. Armato, and F. Lertora, "Design-for-safety for automotive IC design: Challenges and opportunities," in *Proc. of IEEE Custom Integrated Circuit Conference (CICC)*. IEEE, Apr. 2019.
9. *ISO 26262 – Road vehicles – Functional safety*, ISO, 2018. [Online]. Available: https://www.iso.org/standard/68383.html
10. *IEC 61508 – Functional safety - Safety instrumented systems for the process industry sector*, IEC, 2016. [Online]. Available: https://webstore.iec.ch/publication/24241
11. *DO-178C – Software Considerations in Airborne Systems and Equipment Certification*, RTCA, 2012. [Online]. Available: https://www.rtca.org/
12. *DO-254 – Design Assurance Guidance for Airborne Electronic Hardware*, RTCA, 2000. [Online]. Available: https://www.rtca.org/
13. *IEC 62278 – Railway applications - Specification and demonstration of reliability, availability, maintainability and safety (RAMS)*, IEC, 2002. [Online]. Available: https://www.en-standard.eu/iec-62278-2002-railway-applications-specification-and-demonstration-of-reliability-availability-maintainability\-and-safety-rams/
14. *IEC 62061 – Safety of machinery - Functional safety of safety-related control systems*, IEC, 2021. [Online]. Available: https://webstore.iec.ch/publication/59927
15. *UNI EN ISO 13849 – Safety of machinery - Safety-related parts of control systems*, ISO, 2016. [Online]. Available: https://webstore.iec.ch/publication/59927

16. *IEC 61511 – Functional safety - Safety instrumented systems for the process industry sector*, IEC, 2016. [Online]. Available: https://webstore.iec.ch/publication/24241
17. *IEC 61513 – Nuclear power plants - Instrumentation and control important to safety*, IEC, 2011. [Online]. Available: https://webstore.iec.ch/publication/5532
18. *IEC 61513 – Medical device software — Software life cycle processes*, IEC, 2006. [Online]. Available: https://www.iso.org/standard/38421.html
19. J. Wiltgen, "Orchestrating an efficient iso26262 fault campaign," in *White paper*. Siemens, 2021, pp. 1–14. [Online]. Available: https://blogs.sw.siemens.com/verificationhorizons/2021/04/02/orchestrating-an-iso26262-fault-campaign/
20. F. Tao and M. Zhang, "Digital twin shop-floor: A new shop-floor paradigm towards smart manufacturing," *IEEE Access*, vol. 5, pp. 20 418–20 427, 2017.
21. X. Lou, Y. Guo, Y. Gao, K. Waedt, and M. Parekh, "An idea of using digital twin to perform the functional safety and cybersecurity analysis," in *INFORMATIK 2019: 50 Jahre Gesellschaft für Informatik – Informatik für Gesellschaft (Workshop-Beiträge)*, C. Draude, M. Lange, and B. Sick, Eds. Bonn: Gesellschaft für Informatik e.V., 2019, pp. 283–294.
22. F. Tao, M. Zhang, and A. Nee, Eds., *Digital Twin Driven Smart Manufacturing*. Academic Press, 2019.
23. F. Biesinger and M. Weyrich, "The facets of digital twins in production and the automotive industry," in *Proc. of ICMT*, 2019, pp. 1–6.
24. Y. Gao, H. Lv, Y. Hou, J. Liu, and W. Xu, "Real-time modeling and simulation method of digital twin production line," in *Proc. of ITAIC*, 2019, pp. 1639–1642.
25. Y. Ma, H. Zhou, H. He, G. Jiao, and S. Wei, "A digital twin-based approach for quality control and optimization of complex product assembly," in *Proc. of AIAM*, 2019, pp. 762–767.
26. D. Mourtzis, E. Vlachou, N. Milas, and G. Dimitrakopoulos, "Energy consumption estimation for machining processes based on real-time shop floor monitoring via wireless sensor networks," *Procedia CIRP*, vol. 57, pp. 637 – 642, 2016.
27. Y. He, Y. Li, T. Wu, and J. W. Sutherland, "An energy-responsive optimization method for machine tool selection and operation sequence in flexible machining job shops," *Journal of Cleaner Production*, vol. 87, pp. 245 – 254, 2015.
28. M. Zhang, Y. Zuo, and F. Tao, "Equipment energy consumption management in digital twin shop-floor: A framework and potential applications," in *International Conference on Networking, Sensing and Control (ICNSC)*, 2018, pp. 1–5.
29. Y. Ran, X. Zhou, P. Lin, Y. Wen, and R. Deng, "A survey of predictive maintenance: Systems, purposes and approaches," 2019.
30. W. Mahnke, S.-H. Leitner, and M. Damm, *OPC Unified Architecture*. Springer Berlin Heidelberg, 2009.
31. "En 13306:2010 standard - maintenance," [online] Available: https://store.uni.com/en-13306-2010, Aug 2010.
32. IEEE, *IEEE-SA Standards Board P2427/D0.13 Draft Standard for Analog Defect Modeling and Coverage*. IEEE, 2019. [Online]. Available: https://standards.ieee.org/project/2427.html
33. S. Abughannam, L. Wu, W. Mueller, C. Scheytt, W. Ecker, and C. Novello, "Fault injection and mixed-level simulation for analog circuits - a case study," in *IEEE ANALOG 2016; 15. ITG/GMM-Symposium*. IEEE, Sep. 2016.
34. S. Sunter, "Efficient analog defect simulation," in *2019 IEEE International Test Conference (ITC)*. IEEE, Nov. 2019. [Online]. Available: https://doi.org/10.1109/itc44170.2019.9000141
35. S. Sunter and P. Sarson, "A/MS benchmark circuits for comparing fault simulation, DFT, and test generation methods," in *2017 IEEE International Test Conference (ITC)*. IEEE, Oct. 2017.
36. I. S. Esqueda and H. J. Barnaby, "A defect-based compact modeling approach for the reliability of CMOS devices and integrated circuits," *Solid-State Electronics*, vol. 91, pp. 81–86, january 2014.
37. Y. Zhou, H. Liu, S. Mu, Z. Chen, and B. Wang, "Short-circuit failure model of sic mosfet including the interface trapped charges," *IEEE Trans. Emerg. Sel. Topics Power Electron.*, vol. 8, no. 1, pp. 90–98, 2020.
38. S. Rashkeev, C. Cirba, D. Fleetwood, R. Schrimpf, S. Witczak, A. Michez, and S. Pantelides, "Physical model for enhanced interface-trap formation at low dose rates," *IEEE Trans. Nucl. Sci.*, vol. 49, no. 6, pp. 2650–2655, 2002.

39. C. Di and J. Jess, “An efficient cmos bridging fault simulator: with spice accuracy,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 9, pp. 1071–1080, 1996.
40. C. Warwick, “In a nutshell: How spice works,” https://www.emcs.org/acstrial/newsletters/summer09/HowSpiceWorks.pdf, 2009.
41. V. B. Litovski, I. V. Litovski, and M. Zwoliński, “Concurrent analogue fault simulation, the equation formulation aspect,” *International Journal of Circuit Theory and Applications*, vol. 32, no. 6, pp. 487–507, 2004.
42. N. Dall’Ora, S. Azam, E. Fraccaroli, A. Alberts, and F. Fummi, “Predictive fault grouping based on faulty ac matrices,” in *2021 24th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*. IEEE, 2021, pp. 11–16.
43. P. Wilson, Y. Kilic, J. Ross, M. Zwolinski, and A. Brown, “Behavioural modelling of operational amplifier faults using vhdl-ams,” in *Proceedings 2002 Design, Automation and Test in Europe Conference and Exhibition*, 2002, pp. 1133–.
44. H.-G. Stratigopoulos and S. Sunter, “Efficient Monte Carlo-based analog parametric fault modelling,” in *2014 IEEE 32nd VLSI Test Symposium (VTS)*. IEEE, Apr. 2014, pp. 1–6.
45. A. Maheshwari, I. Koren, and N. Burleson, “Techniques for transient fault sensitivity analysis and reduction in VLSI circuits,” in *Proceedings 18th IEEE Symposium on Defect and Fault Tolerance in VLSI Systems*, 2003, pp. 597–604.
46. X. Gao, J. Liou, J. Bernier, G. Croft, and A. Ortiz-Conde, “Implementation of a comprehensive and robust mosfet model in cadence spice for esd applications,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, no. 12, pp. 1497–1502, 2002.
47. M. Eslami, B. Ghavami, M. Raji, and A. Mahani, “A survey on fault injection methods of digital integrated circuits,” *Integration*, vol. 71, pp. 154–163, mar 2020.
48. J. A. S. Augusto, “Efficient time domain analogue fault simulation targeting nonlinear circuits,” in *2011 20th European Conference on Circuit Theory and Design (ECCTD)*. IEEE, Aug. 2011.
49. N. Engin and H. Kerkhoff, “Fast fault simulation for nonlinear analog circuits,” *IEEE Design & Test of Computers*, vol. 20, no. 2, pp. 40–47, Mar. 2003.
50. B. Tasić, J. J. Dohmen, R. Janssen, E. J. W. ter Maten, R. Pulch, and T. G. J. Beelen, “Fast time-domain simulation for reliable fault detection,” in *Proceedings of the 2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. Research Publishing Services, 2016, pp. 301–306.
51. L. Xia, M. U. Farooq, I. M. Bell, F. A. Hussin, and A. S. Malik, “Survey and Evaluation of Automated Model Generation Techniques for High Level Modeling and High Level Fault Modeling,” *Journal of Electronic Testing*, vol. 29, no. 6, pp. 861–877, Aug. 2013.
52. M. Zwolinski and A. Brown, “Behavioural modelling of analogue faults in VHDL-AMS - a case study,” in *2004 IEEE International Symposium on Circuits and Systems (IEEE Cat. No.04CH37512)*, vol. 5, 2004, pp. V–V.
53. Y. Kiliç and M. Zwoliński, “Behavioral fault modeling and simulation using VHDL-AMS to speed-up analog fault simulation,” *Analog Integrated Circuits and Signal Processing*, vol. 39, no. 2, pp. 177–190, may 2004.
54. S.-N. Ahmadian and S.-G. Miremadi, “Fault injection in mixed-signal environment using behavioral fault modeling in verilog-a,” in *2010 IEEE International Behavioral Modeling and Simulation Workshop*, 2010, pp. 69–74.
55. P. Wilson, Y. Kilic, J. Ross, M. Zwolinski, and A. Brown, “Behavioural modelling of operational amplifier faults using analogue hardware description languages,” in *Proceedings of the Fifth IEEE International Workshop on Behavioral Modeling and Simulation. BMAS 2001 (Cat No.01TH8601)*. IEEE, 2001, pp. 106–112.
56. W. Zheng, Y. Feng, X. Huang, and H. Mantooth, “Ascend: automatic bottom-up behavioral modeling tool for analog circuits,” in *2005 IEEE International Symposium on Circuits and Systems*. IEEE, 2005, pp. 5186–5189 Vol. 5.
57. S. Puthiyottil and E. Sureshkumar, “Speed Enhanced Mixed Signal Design-for-Test Using Hybrid Fault Based Testing Algorithms,” in *2010 Second International Conference on Computer Modeling and Simulation*, vol. 4. IEEE, Jan. 2010, pp. 270–274.
58. J. Parky, S. Madhavapeddiz, A. Paglieri, C. Barrz, and J. A. Abrahamy, “Defect-based analog fault coverage analysis using mixed-mode fault simulation,” in *2009 IEEE 15th International Mixed-Signals, Sensors, and Systems Test Workshop*. IEEE, Jun. 2009, pp. 1–6.

59. L. Fang, G. Gronthoud, and H. Kerkhoff, “Reducing analogue fault-simulation time by using ifigh-level modelling in dotss for an industrial design,” in *IEEE European Test Workshop, 2001.* IEEE, 2001, pp. 61–67.
60. X. Li, X. Zeng, D. Zhou, and X. Ling, “Behavioral modeling of analog circuits by wavelet collocation method,” in *IEEE/ACM International Conference on Computer Aided Design. ICCAD 2001. IEEE/ACM Digest of Technical Papers (Cat. No.01CH37281).* IEEE, 2001, pp. 65–69.
61. D. Shaw, D. Al-Khalili, and C. Rozon, “Automatic generation of defect injectable VHDL fault models for ASIC standard cell libraries,” *Integration*, vol. 39, no. 4, pp. 382–406, Jul. 2006.
62. N. Zain Ali, M. Zwolinski, and A. Ahmadi, “Delay fault modelling/simulation using VHDL-AMS in multi-Vdd systems,” in *2008 26th International Conference on Microelectronics*, 2008, pp. 413–416.
63. E. Romero, G. Peretti, and C. Marqués, “An operational amplifier model for evaluating test strategies at behavioural level,” *Microelectronics Journal*, vol. 38, no. 10-11, pp. 1082–1094, Oct. 2007.
64. B. Straube, W. Vermeiren, and V. Spenke, “Multi-level hierarchical analogue fault simulation,” *Microelectronics Journal*, vol. 33, no. 10, pp. 815–821, Oct. 2002.
65. F. Liu and S. Ozev, “Efficient simulation of parametric faults for multi-stage analog circuits,” in *2007 IEEE International Test Conference*, 2007, pp. 1–9.
66. J. Hou and A. Chatterjee, “Analog transient concurrent fault simulation with dynamic fault grouping,” in *Proceedings 2000 International Conference on Computer Design.* IEEE Comput. Soc, Sep 2000, pp. 35–41.
67. S. Manetti and M. Piccirilli, “A singular-value decomposition approach for ambiguity group determination in analog circuits,” *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 50, no. 4, pp. 477–487, 2003.
68. M. Worsman, M. Wong, and Y. Lee, “Enhancing the fault diagnosis of linear analog circuit steady-state DC testing through the analysis of equivalent faults,” *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 50, no. 7, pp. 932–936, 2003.
69. O. Karaca, J. Kirscher, A. Laroche, A. Tributsch, L. Maurer, and G. Pelz, “Fault grouping for fault injection based simulation of ams circuits in the context of functional safety,” in *2016 13th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*, 2016, pp. 1–4.
70. R. Martins, N. Lourenço, N. Horta, N. Guerreiro, and M. Santos, “Embedding Fault List Compression techniques in a design automation framework for analog and Mixed-Signal structural testing,” in *2015 Conference on Design of Circuits and Integrated Systems (DCIS)*, 2015, pp. 1–6.
71. N. Guerreiro and M. Santos, “Mixed-Signal Fault Equivalence: Search and Evaluation,” *2011 Asian Test Symposium*, pp. 377–382, 2011.
72. O. Karaca, J. Kirscher, A. Laroche, A. Tributsch, L. Maurer, and G. Pelz, “Fault grouping for fault injection based simulation of AMS circuits in the context of functional safety,” in *2016 13th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD).* IEEE, Jun 2016, pp. 1–4.
73. N. Guerreiro, M. Santos, and P. Teixeira, “Fault list compression for efficient analogue and mixed-signal production test preparation,” in *Design of Circuits and Integrated Systems.* IEEE, Nov 2014, pp. 1–6.
74. K. Huang, H. G. Stratigopoulos, S. Mir, C. Hora, Y. Xing, and B. Kruseman, “Diagnosis of Local Spot Defects in Analog Circuits,” *IEEE Transactions on Instrumentation and Measurement*, vol. 61, no. 10, pp. 2701–2712, Oct. 2012.
75. N. Guerreiro, M. Santos, and P. Teixeira, “Analogue and Mixed-Signal Production Test Speed-Up by Means of Fault List Compression,” *Circuits and Systems*, vol. 04, no. 05, pp. 407–421, Sep. 2013.
76. M. Worsman, M. Wong, and Y. Lee, “Analog circuit equivalent faults in the D.C. domain,” in *Proceedings of the Ninth Asian Test Symposium*, 2000, pp. 84–89.
77. Y. Maidon, T. Zimmer, and A. Ivanov, “An Analog Circuit Fault Characterization Methodology,” *Journal of Electronic Testing*, vol. 21, no. 2, pp. 127–134, Apr. 2005.
78. Y.-H. Chang, “Frequency-domain grouping robust fault diagnosis for analog circuits with uncertainties,” *International Journal of Circuit Theory and Applications*, vol. 30, no. 1, pp. 65–86, Jan 2002.

79. S. Sanyal, S. P. P. K. Garapati, A. Patra, P. Dasgupta, and M. Bhattacharya, “Fault classification and coverage of analog circuits using DC operating point and frequency response analysis,” in *Proceedings of the 2019 on Great Lakes Symposium on VLSI*. ACM, May 2019, pp. 123—-128.
80. J. Hou and A. Chatterjee, “Concurrent transient fault simulation for analog circuits,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 10, pp. 1385–1398, Oct. 2003.
81. K. Saab, N. Hamida, and B. Kaminska, “Closing the gap between analog and digital testing,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 2, pp. 307–314, 2001.
82. B. Tasic, J. J. Dohmen, E. J. W. ter Maten, T. G. Beelen, W. H. Schilders, A. de Vries, and M. van Beurden, “Robust DC and efficient time-domain fast fault simulation,” *COMPEL: The International Journal for Computation and Mathematics in Electrical and Electronic Engineering*, vol. 33, no. 4, pp. 1161–1174, Jul. 2014.
83. S. Sunter, K. Jurga, P. Dingenen, and R. Vanhooren, “Practical random sampling of potential defects for analog fault simulation,” in *2014 International Test Conference*. IEEE, Oct. 2014, pp. 1–10.
84. D. De Jonghe, E. Maricau, G. Gielen, T. McConaghy, B. Tasić, and H. Stratigopoulos, “Advances in variation-aware modeling, verification, and testing of analog ICs,” in *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2012, pp. 1615–1620.
85. S. Mosin, “Automated simulation of faults in analog circuits based on parallel paradigm,” in *2017 IEEE East-West Design Test Symposium (EWDTS)*, 2017, pp. 1–6.
86. E. Schneider and H.-J. Wunderlich, “High-Throughput Transistor-Level Fault Simulation on GPUs,” in *2016 IEEE 25th Asian Test Symposium (ATS)*, 2016, pp. 150–155.
87. B. Lapeyre and J. Lelong, “A framework for adaptive Monte Carlo procedures,” *Monte Carlo Methods and Applications*, vol. 17, no. 1, Jan. 2011.
88. K. Saab, N. Ben-Hamida, and B. Kaminska, “Parametric fault simulation and test vector generation,” in *Proceedings Design, Automation and Test in Europe Conference and Exhibition 2000 (Cat. No. PR00537)*. IEEE Comput. Soc, 2000, pp. 650–656.
89. S. Sunter, K. Jurga, and A. Laidler, “Using Mixed-Signal Defect Simulation to Close the Loop Between Design and Test,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 63, no. 12, pp. 2313–2322, dec 2016.
90. B. Jourdain and J. Lelong, “Robust adaptive importance sampling for normal random vectors,” *The Annals of Applied Probability*, vol. 19, no. 5, pp. 1687–1718, Oct. 2009.
91. E. M. ter, T. Doorn, J. Croon, A. Bargagli, A. Di Bucchianico, and O. Wittich, “Importance sampling for high speed statistical Monte-Carlo simulations,” in *NXP Technical Note NXP-TN-2007-00238, NXP Semi-conductors*, 2007.
92. H.-G. Stratigopoulos and S. Sunter, “Fast Monte Carlo-Based Estimation of Analog Parametric Test Metrics,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 12, pp. 1977–1990, 2014.
93. E. Yilmaz, G. Shofner, L. Winemberg, and S. Ozev, “Fault Analysis and Simulation of Large Scale Industrial Mixed-Signal Circuits,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013*. IEEE Conference Publications, May 2013, pp. 565–570.
94. S. Sunter, “Experiences with an industrial analog fault simulator and engineering intuition,” in *2015 IEEE 20th International Mixed-Signals Testing Workshop (IMSTW)*, 2015, pp. 1–5.
95. L. Ji and X. Hu, “Analog circuit soft-fault diagnosis based on sensitivity analysis with minimum fault number rule,” *Analog Integrated Circuits and Signal Processing*, vol. 95, no. 1, pp. 163–171, Jan. 2018.
96. M. Singh, R. Rachala, and I. Koren, “Transient fault sensitivity analysis of analog-to-digital converters (ADCs),” in *Proceedings IEEE Computer Society Workshop on VLSI 2001. Emerging Technologies for VLSI Systems*. IEEE Computer Society, 2001, pp. 140–145.
97. M. Singh and I. Koren, “Fault-sensitivity analysis and reliability enhancement of analog-to-digital converters,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 11, no. 5, pp. 839–852, 2003.
98. H. Hashempour, J. Dohmen, B. Tasić, B. Kruseman, C. Hora, M. van Beurden, and Y. Xing, “Test time reduction in analogue/mixed-signal devices by defect oriented testing: An industrial example,” in *2011 Design, Automation Test in Europe*, 2011, pp. 1–6.

99. C.-J. Shi, M. Tian, and G. Shi, "Efficient DC fault simulation of nonlinear analog circuits: one-step relaxation and adaptive simulation continuation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 7, pp. 1392–1400, 2006.
100. J. Brenkuš, V. Stopjaková, and G. Gyepes, "Numerical method for DC fault analysis simplification and simulation time reduction," in *2013 IEEE 16th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*, 2013, pp. 170–174.
101. E. Fraccaroli, M. Lora, and F. Fummi, "Automatic abstraction of multi-discipline analog models for efficient functional simulation," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, Mar. 2017.
102. E. Fraccaroli and F. Fummi, "Analog fault testing through abstraction," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, 2017, pp. 270–273.
103. E. Fraccaroli, D. Quaglia, and F. Fummi, "Simulation-based Holistic Functional Safety Assessment for Networked Cyber-Physical Systems," in *2018 Forum on Specification & Design Languages (FDL)*, 2018, pp. 5–16.
104. Z. Liu, S. K. Chaganti, and D. Chen, "Improving Time-Efficiency of Fault-Coverage Simulation for MOS Analog Circuit," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 5, pp. 1664–1674, May 2018.
105. M. Tadeusiewicz and S. Hałgas, "A method for fast simulation of multiple catastrophic faults in analogue circuits," *International Journal of Circuit Theory and Applications*, pp. n/a–n/a, 2008.
106. E. Yilmaz, A. Meixner, and S. Ozev, "An industrial case study of analog fault modeling," in *29th VLSI Test Symposium*. IEEE, May 2011, pp. 178–183.
107. W. Scherr and K. Einwich, "Beyond real number modeling: Comparison of analog modeling approaches," in *2020 Forum for Specification and Design Languages (FDL)*. IEEE, Sep. 2020.
108. A. Vladimirescu and J.-J. Charlot, "Challenges of mos analog circuit simulation with spice," in *IEE Colloquium on SPICE: Surviving Problems in Circuit Evaluation*, 1993, pp. 9/1–9/5.
109. R. Pratap, V. Agarwal, and R. K. Singh, "Review of various available spice simulators," in *2014 International Conference on Power, Control and Embedded Systems (ICPCES)*, 2014, pp. 1–6.
110. K. S. Kundert and P. Gray, *The Designer's Guide to Spice and Spectre*. USA: Kluwer Academic Publishers, 1995.
111. N. Dall'Ora, S. Azam, E. Fraccaroli, A. Alberts, and F. Fummi, "A common manipulation framework for transistor-level languages," in *2021 Forum on specification & Design Languages (FDL)*. IEEE, 2021, pp. 01–07.
112. F. Pecheux, C. Lallement, and A. Vachoux, "VHDL-AMS and verilog-AMS as alternative hardware description languages for efficient modeling of multidiscipline systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 2, pp. 204–225, 2005.
113. "Spice users guide," https://irsa.ipac.caltech.edu/data/SPITZER/docs/data analysistools/tools/spice/spiceusersguide/SPICE_Users_Guide.pdf, v.2.3.1.
114. "Pspice users guide," https://electrical.engineering.unt.edu/sites/default/files/Cadence_PSpice.pdf, 2000.
115. Synopsys, "Hspice user guide revision 2008," https://cseweb.ucsd.edu/classes/wi10/cse241a/assign/hspice_sa.pdf, 2008.
116. "Spectre circuit simulator reference manual," http://web.engr.uky.edu/ elias/tutorials/Spectre/spectre_refManual.pdf, 2020.
117. "Verilog-AMS Language Reference Manual," 2014. [Online]. Available: https://www.accellera.org/images/downloads/standards/v-ams/VAMS-LRM-2-4.pdf
118. Siemens, "Eldo platform language reference manual revision 2005," http://web.engr.uky.edu/~elias/tutorials/Eldo/eldo_ur.pdf, 2005.
119. A. Technologies, "Netlist translator for spice and spectre," https://pages.jh.edu/aandreo1/495/Archives/Bibliography/CAD/SPICE_Models.5Equivalency.pdf, 2002.
120. D. Batas and H. Fiedler, "A python interface for spice-based simulations," in *ICSES 2010 International Conference on Signals and Electronic Circuits*, 2010, pp. 161–164.
121. S. Sunter, K. Jurga, P. Dingenen, and R. Vanhooren, "Practical random sampling of potential defects for analog fault simulation," in *2014 International Test Conference*, 2014, pp. 1–10.

122. S. Sunter, “Analog fault simulation - a hot topic!” in *2020 IEEE European Test Symposium (ETS)*, 2020, pp. 1–5.
123. Y. Kiliç and M. Zwoliński, “Behavioral fault modeling and simulation using VHDL-AMS to speed-up analog fault simulation,” *Analog Integrated Circuits and Signal Processing*, vol. 39, no. 2, pp. 177–190, May 2004.
124. E. Yilmaz, A. Meixner, and S. Ozev, “An industrial case study of analog fault modeling,” in *29th VLSI Test Symposium*, 2011, pp. 178–183.
125. B. Esen, A. Coyette, G. Gielen, W. Dobbelaere, and R. Vanhooren, “Effective dc fault models and testing approach for open defects in analog circuits,” in *2016 IEEE International Test Conference (ITC)*, 2016, pp. 1–9.
126. E. Yilmaz, G. Shofner, L. Winemberg, and S. Ozev, “Fault analysis and simulation of large scale industrial mixed-signal circuits,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013*. IEEE Conference Publications, May 2013, pp. 565–570.
127. K. Jung, W. Eisenstadt, and R. Fox, “SPICE-based mixed-mode s-parameter calculations for four-port and three-port circuits,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 5, pp. 909–913, May 2006.
128. S.-S. Lu, T.-W. Chen, H.-C. Chen, and C.-C. Meng, “A novel interpretation of transistor s-parameters by poles and zeros for RF IC circuit design,” *IEEE Transactions on Microwave Theory and Techniques*, vol. 49, no. 2, pp. 406–409, 2001.
129. Y.-J. Kim, O.-K. Kwon, and C.-H. Lee, “Equivalent circuit extraction from the measured s-parameters of electronic packages,” in *ICVC '99. 6th International Conference on VLSI and CAD (Cat. No.99EX361)*. IEEE, Aug 1999, pp. 415–418.
130. J.-C. G. Santos and R. Torres-Torres, “Assessing the accuracy of the open, short and open-short de-embedding methods for on-chip transmission line s-parameters measurements,” in *2017 International Caribbean Conference on Devices, Circuits and Systems (ICCDCS)*. IEEE, Jun 2017, pp. 57–60.
131. G. Leger and A. Gines, “Likelihood-sampling adaptive fault simulation,” in *2017 International Mixed Signals Testing Workshop (IMSTW)*, 2017, pp. 1–6.
132. S. Sunter, K. Jurga, P. Dingenen, and R. Vanhooren, “Practical random sampling of potential defects for analog fault simulation,” in *2014 International Test Conference*, 2014, pp. 1–10.
133. M. W. Tian and C.-J. R. Shi, “Rapid frequency-domain analog fault simulation under parameter tolerances,” in *Proceedings of the 34th annual conference on Design automation conference - DAC '97*. ACM Press, Aug 1997, pp. 275–280.
134. C. Tenorio de Carvalho and L. de Menezes, “Extraction of s-parameter using the three-dimensional transmission-line matrix (tlm) method,” in *Proc. of the 2003 SBMO/IEEE MTT-S Int. Microwave and Optoelectronics Conference - IMOC 2003. (Cat. No.03TH8678)*, vol. 2, 2003, pp. 967–969.
135. S. Tian, C. Yang, F. Chen, and Z. Liu, “Circle equation-based fault modeling method for linear analog circuits,” *IEEE Transactions on Instrumentation and Measurement*, vol. 63, no. 9, pp. 2145–2159, Sep. 2014.
136. A. Al-Sharadqah and N. Chernov, “Error analysis for circle fitting algorithms,” *Electronic Journal of Statistics*, vol. 3, no. 0, pp. 886–911, Jul 2009.
137. K. Zhang, Y. Shi, S. Karnouskos, T. Sauter, H. Fang, and A. W. Colombo, “Advancements in industrial cyber-physical systems: An overview and perspectives,” *IEEE Transactions on Industrial Informatics*, vol. 19, no. 1, pp. 716–729, 2023.
138. L. D. Lago, O. Ferrante, R. Passerone, and A. Ferrari, “Dependability assessment of SOA-based CPS with contracts and model-based fault injection,” *IEEE Transactions on Industrial Informatics*, vol. 14, no. 1, pp. 360–369, jan 2018.
139. P. Gardonio and M. J. Brennan, “On the origins and development of mobility and impedance methods in structural dynamics,” *Journal of Sound and vibration*, vol. 249, no. 3, pp. 557–573, 2002.
140. Z. Li, Y. Wang, and K. Wang, “A deep learning driven method for fault classification and degradation assessment in mechanical equipment,” *Computers in Industry*, vol. 104, pp. 1–10, 2019.
141. L. M. R. Baccarini, V. V. R. e Silva, B. R. de Menezes, and W. M. Caminhas, “Svm practical industrial application for mechanical faults diagnostic,” *Expert Systems with Applications*, vol. 38, no. 6, pp. 6980–6984, 2011.

142. S. Centomo, N. Dall'Ora, and F. Fummi, "The design of a digital-twin for predictive maintenance," in *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1. IEEE, 2020, pp. 1781–1788.
143. F. Pecheux, C. Lallement, and A. Vachoux, "Vhdl-ams and verilog-ams as alternative hardware description languages for efficient modeling of multidiscipline systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 2, pp. 204–225, 2005.
144. P. Frey and D. O'Riordan, "Verilog-AMS: Mixed-signal simulation and cross domain connect modules," in *Proc. of IEEE/ACM International Workshop on Behavioral Modeling and Simulation*. IEEE Comput. Soc, 2000.
145. C. Chen *et al.*, "Research on mechanical fault injection method based on ADAMS," in *2019 IEEE 3rd Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*. IEEE, Oct. 2019.
146. S. J. Uder, R. B. Stone, and I. Y. Tumer, "Failure analysis in subsystem design for space missions," in *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, vol. 46962, 2004, pp. 201–217.
147. A. Kolesnikov, D. Tretsiak, and M. Cameron, "Systematic simulation of fault behavior by analysis of vehicle dynamics," in *Proceedings of the 13th International Modelica Conference, Regensburg, Germany, March 4–6, 2019*. Linköing University Electronic Press, Feb. 2019.
148. J. Gundermann, A. Kolesnikov, M. Cameron, and T. Blochwitz, "The fault library - a new modelica library allows for the systematic simulation of non-nominal system behavior," in *Proc. of the 2nd Japanese Modelica Conference Tokyo*, 2019.
149. M. Jiménez-Guarneros, C. Morales-Perez, and J. d. J. Rangel-Magdaleno, "Diagnostic of combined mechanical and electrical faults in asd-powered induction motor using modwt and a lightweight 1-d cnn," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 7, pp. 4688–4697, 2022.
150. T. Pan, J. Chen, J. Xie, Z. Zhou, and S. He, "Deep feature generating network: A new method for intelligent fault detection of mechanical systems under class imbalance," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 9, pp. 6282–6293, 2021.
151. E. Balaban, P. Bansal, P. Stoelting, A. Saxena, K. F. Goebel, and S. Curran, "A diagnostic approach for electro-mechanical actuators in aerospace systems," in *2009 IEEE Aerospace conference*. IEEE, 2009, pp. 1–13.
152. R. Mariani and G. Boschi, "A systematic approach for failure modes and effects analysis of system-on-chips," in *13th IEEE International On-Line Testing Symposium (IOLTS 2007)*, 2007, pp. 187–188.
153. A. Baklouti, N. Nguyen, F. Mhenni, J.-Y. Choley, and A. Mlika, "Improved safety analysis integration in a systems engineering approach," *Applied Sciences*, vol. 9, no. 6, 2019. [Online]. Available: https://www.mdpi.com/2076-3417/9/6/1246
154. J. Wu and S. Yan, "Fault severity evaluation and improvement design for mechanical systems using the fault injection technique and gini concordance measure," *Mathematical Problems in Engineering*, vol. 2014, pp. 1–20, 2014.
155. N. Bombieri, G. Di Guglielmo, M. Ferrari, F. Fummi, G. Pravadelli, F. Stefanni, and A. Venturelli, "Hifsuite: tools for hdl code conversion and manipulation," *EURASIP Journal on Embedded Systems*, vol. 2010, no. 1, pp. 1–20, 2010.
156. S. Vinco, V. Guarnieri, and F. Fummi, "Code Manipulation for Virtual Platform Integration," *IEEE Transactions on Computers*, vol. 65, no. 9, pp. 2694–2708, 2016.
157. M. Lora, S. Vinco, E. Fraccaroli, D. Quaglia, and F. Fummi, "Analog models manipulation for effective integration in smart system virtual platforms," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 2, pp. 378–391, feb 2018.
158. S. Centomo, M. Lora, A. Portaluri, F. Stefanni, and F. Fummi, "Automatic integration of HDL IPs in simulink using FMI and s-function interfaces," in *Lecture Notes in Electrical Engineering*. Springer International Publishing, dec 2018, pp. 1–23.
159. D. Shin, M. Poncino, E. Macii, and N. Chang, "A statistical model-based cell-to-cell variability management of li-ion battery pack," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 2, pp. 252–265, 2015.

160. R. R. Thakkar, “Electrical equivalent circuit models of lithium-ion battery,” in *Energy Storage Devices [Working Title]*. IntechOpen, Sep. 2021.
161. F. Grossi, A. Palladino, R. Zanasi, and G. Fiengo, “The pog technique for modelling engine dynamics based on electrical analogy,” in *2009 American Control Conference*, 2009, pp. 2057–2063.
162. K. Minami and Y. Kajikawa, “A study on frequency response analysis of micro-speaker using equivalent circuit and finite element method,” in *2020 IEEE 9th Global Conference on Consumer Electronics (GCCE)*, 2020, pp. 723–725.
163. C. Campbell, “Equivalent circuit models for a SAW filter,” in *Surface Acoustic Wave Devices and their Signal Processing Applications*. Elsevier, 1989, pp. 59–83.
164. T. Veijola, “An electrical equivalent circuit model for rf mems disk resonators,” in *2007 18th European Conference on Circuit Theory and Design*, 2007, pp. 962–965.
165. T. Fukazawa and Y. Tanaka, “Spontaneous otoacoustic emissions in an active feed-forward model of the cochlea,” *Hearing Research*, vol. 95, no. 1, pp. 135–143, 1996.
166. A. Bloch, “Electromechanical analogies and their use for the analysis of mechanical and electromechanical systems,” *Journal of the Institution of Electrical Engineers - Part I: General*, vol. 92, no. 52, pp. 157–169, 1945.
167. F. A. Firestone, “A NEW ANALOGY BETWEEN MECHANICAL AND ELECTRICAL SYSTEMS,” *The Journal of the Acoustical Society of America*, vol. 4, no. 3, pp. 249–267, Jan. 1933. [Online]. Available: https://doi.org/10.1121/1.1915605
168. A. C. Fairlie-Clarke, “Force as a flow variable,” *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, vol. 213, no. 1, pp. 77–81, 1999.
169. T. Blochwitz *et al.*, “Functional mockup interface 2.0: The standard for tool independent exchange of simulation models,” in *Proc. of MODELICA Conference 2012*, 2012, pp. 173–184.
170. J. López-Martínez, J. C. Martínez, D. García-Vallejo, A. Alcayde, and F. G. Montoya, “A new electromechanical analogy approach based on electrostatic coupling for vertical dynamic analysis of planar vehicle models,” *IEEE Access*, vol. 9, pp. 119 492–119 502, 2021.
171. D. Jeltsema and J. M. Scherpen, “Multidomain modeling of nonlinear networks and systems,” *IEEE Control Systems Magazine*, vol. 29, no. 4, pp. 28–59, 2009.
172. M. Nagurka and S. Huang, “A mass-spring-damper model of a bouncing ball,” in *Proceedings of the 2004 American Control Conference*, vol. 1, 2004, pp. 499–504 vol.1.
173. E. Shahabpoor, A. Pavic, and V. Racic, “Identification of mass–spring–damper model of walking humans,” *Structures*, vol. 5, pp. 233–246, 2016.
174. C. A. Jones, A. Pavic, P. Reynolds, and R. E. Harrison, “Verification of equivalent mass–spring–damper models for crowd–structure vibration response prediction,” *Canadian Journal of Civil Engineering*, vol. 38, no. 10, pp. 1122–1135, 2011.
175. M. Smith, “Synthesis of mechanical networks: the inerter,” *IEEE Transactions on Automatic Control*, vol. 47, no. 10, pp. 1648–1662, 2002.
176. J. López-Martínez, J. Martínez, D. García-Vallejo, A. Alcayde, and F. G. Montoya, “A new electromechanical analogy approach based on electrostatic coupling for vertical dynamic analysis of planar vehicle models,” 2020.
177. N. Dall’Ora, S. Vinco, and F. Fummi, “Functionality and fault modeling of a dc motor with verilog-ams,” in *2020 IEEE 18th International Conference on Industrial Informatics (INDIN)*, vol. 1. IEEE, 2020, pp. 35–40.
178. N. Dall’Ora, E. Fraccaroli, S. Vinco, and F. Fummi, “Multi-discipline fault modeling with verilog-ams,” in *2021 4th IEEE International Conference on Industrial Cyber-Physical Systems (ICPS)*, 2021, pp. 237–243.
179. C. Chen, Q. Shi, F. Zhang, and Z. You, “Research on mechanical fault injection method based on adams,” in *2019 IEEE 3rd Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, 2019, pp. 418–421.
180. S.-H. Kim, “Control of direct current motors,” in *Electric Motor Control*. Elsevier, 2017, pp. 39–93.
181. K. Kundert and O. Zinke, *The designer’s guide to Verilog-AMS*. Springer Science & Business Media, 2006.
182. S.-H. Yen, P.-C. Tang, Y.-C. Lin, and C.-Y. Lin, “A sensorless and low-gain brushless DC motor controller using a simplified dynamic force compensator for robot arm application,” *Sensors*, vol. 19, no. 14, p. 3171, Jul. 2019.

183. M. Lora, S. Centomo, D. Quaglia, and F. Fummi, “Automatic integration of cycle-accurate descriptions with continuous-time models for cyber-physical virtual platforms,” in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, Mar. 2018. [Online]. Available: https://doi.org/10.23919/date.2018.8342095
184. M. Lora, S. Vinco, E. Fraccaroli, D. Quaglia, and F. Fummi, “Analog models manipulation for effective integration in smart system virtual platforms,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 2, pp. 378–391, 2018.
185. S. Vinco, Y. Chen, F. Fummi, E. Macii, and M. Poncino, “A layered methodology for the simulation of extra-functional properties in smart systems,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 10, pp. 1702–1715, 2017.
186. T. Yaren, S. Kizir, and A. B. Yildiz, “Electrical equivalent circuit based analysis of double pendulum system,” in *2019 6th International Conference on Electrical and Electronics Engineering (ICEEE)*, 2019, pp. 258–262.
187. M. Hassan, D. Große, H. M. Le, T. Vörtler, K. Einwich, and R. Drechsler, “Testbench qualification for systemc-ams timed data flow models,” in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2018, pp. 857–860.
188. N. Dall’Ora, S. Centomo, and F. Fummi, “Industrial-iot data analysis exploiting electronic design automation techniques,” in *2019 IEEE 8th International Workshop on Advances in Sensors and Interfaces (IWASI)*, 2019, pp. 103–109.
189. S. Vinco, Y. Chen, F. Fummi, E. Macii, and M. Poncino, “A layered methodology for the simulation of extra-functional properties in smart systems,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 10, pp. 1702–1715, oct 2017.
190. N. Hogan and P. C. Breedveld, “The physical basis of analogies in physical system models,” in *The mechatronics handbook*. CRC Press/Balkema, 2002.
191. C.-A. Chen, X. Li, L. Zuo, and K. D. Ngo, “Circuit modeling of the mechanical-motion rectifier for electrical simulation of ocean wave power takeoff,” *IEEE Transactions on Industrial Electronics*, vol. 68, no. 4, pp. 3262–3272, 2020.
192. B. C. Guo, Y. K. Huang, Y. G. Guo, and J. G. Zhu, “Thermal analysis of the conical rotor motor using lptn combined with fluid simulation,” in *2015 IEEE International Conference on Applied Superconductivity and Electromagnetic Devices (ASEMD)*, 2015, pp. 128–129.
193. J. Klink, J. Grabow, N. Orazov, R. Benger, A. Börger, A. Ahlberg Tidblad, H. Wenzl, and H.-P. Beck, “Thermal fault detection by changes in electrical behaviour in lithium-ion cells,” *Journal of Power Sources*, vol. 490, 2021.
194. Y. Chen, S. Vinco, E. Macii, and M. Poncino, “Fast thermal simulation using systemc-ams,” in *Proc. of the Great Lakes Symposium on VLSI*, 2016, p. 427–432.
195. K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, “Temperature-aware microarchitecture,” *SIGARCH Comput. Archit. News*, vol. 31, no. 2, p. 2–13, 2003.
196. N. An, M. Du, Z. Hu, and K. Wei, “A high-precision adaptive thermal network model for monitoring of temperature variations in insulated gate bipolar transistor (igbt) modules,” *Energies*, vol. 11, no. 3, p. 595, 2018.
197. O. Alavi, M. Abdollah, and A. H. Viki, “Assessment of thermal network models for estimating igbt junction temperature of a buck converter,” in *2017 8th Power Electronics, Drive Systems & Technologies Conference (PEDSTC)*. IEEE, 2017, pp. 102–107.
198. D. Troncon and L. Alberti, “Case of study of the electrification of a tractor: Electric motor performance requirements and design,” *Energies*, vol. 13, no. 9, p. 2197, 2020.
199. G. Swift, T. S. Molinski, and W. Lehn, “A fundamental approach to transformer thermal modeling. i. theory and equivalent circuit,” *IEEE transactions on Power Delivery*, vol. 16, no. 2, pp. 171–175, 2001.
200. K. Murthy and R. Bedford, “Transformation between foster and cauer equivalent networks,” *IEEE Transactions on Circuits and Systems*, vol. 25, no. 4, pp. 238–239, 1978.
201. L. Mo, T. Zhang, and Q. Lu, “Thermal analysis of a flux-switching permanent-magnet double-rotor machine with a 3-d thermal network model,” *IEEE Transactions on Applied Superconductivity*, vol. 29, no. 2, pp. 1–5, 2019.

202. K. Nishi, T. Hatakeyama, S. Nakagawa, and M. Ishizuka, "Transient thermal analysis of the microprocessor system one-dimensional thermal network with power estimation equation," in *Fourteenth Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm)*, 2014, pp. 982–989.
203. F. Tosoni, N. Dall'Ora, E. Fraccaroli, and F. Fummi, "A framework for modeling and concurrently simulating mechanical and electrical faults in verilog-ams," in *2022 Forum on Specification & Design Languages (FDL)*. IEEE, 2022, pp. 1–8.
204. J. W. Li, W. J. Zhang, G. S. Yang, S. D. Tu, and X. B. Chen, "Thermal-error modeling for complex physical systems: the-state-of-arts review," *The International Journal of Advanced Manufacturing Technology*, vol. 42, no. 1-2, pp. 168–179, Jun. 2008.
205. N. Dall'Ora, F. Tosoni, E. Fraccaroli, and F. Fummi, "Inferring mechanical fault models from the electrical domain," in *2022 IEEE 5th International Conference on Industrial Cyber-Physical Systems (ICPS)*. IEEE, 2022, pp. 01–08.
206. L.-R. Zheng, M. Shen, X. Duo, and H. Tenhunen, "Embedded smart systems for intelligent paper and packaging," in *Proceedings Electronic Components and Technology, 2005. ECTC '05.*, 2005, pp. 1776–1782 Vol. 2.
207. "Cadence virtual system platform," https://www.cadence.com/ko_KR/home /tools/system-design-and-verification/software-driven-verification/virtual-system-platform.html, 2021.
208. J. L. V. S. de la Vega and E. Tlelo-Cuautle, "Simulation of piecewise-linear one-dimensional chaotic maps by verilog-a," *IETE Technical Review*, vol. 32, no. 4, pp. 304–310, Mar. 2015.
209. Y. Zhang, S. Sankaranarayanan, and F. Somenzi, "Piecewise linear modeling of nonlinear devices for formal verification of analog circuits," in *2012 Formal Methods in Computer-Aided Design (FMCAD)*, 2012, pp. 196–203.
210. J. Goncalves, A. Megretski, and M. Dahleh, "Global analysis of piecewise linear systems using impact maps and surface lyapunov functions," *IEEE Transactions on Automatic Control*, vol. 48, no. 12, pp. 2089–2106, 2003.
211. M. Lora, S. Vinco, E. Fraccaroli, D. Quaglia, and F. Fummi, "Analog models manipulation for effective integration in smart system virtual platforms," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 2, pp. 378–391, 2018.
212. A. Tarraf and L. Hedrich, "Behavioral modeling of transistor-level circuits using automatic abstraction to hybrid automata," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, Mar. 2019.
213. E. Tlelo-Cuautle and M. Duarte-Villaseñor, "Designing chua's circuit from the behavioral to the transistor level of abstraction," *Applied Mathematics and Computation*, vol. 184, no. 2, pp. 715–720, Jan. 2007.
214. S. Dam and P. Mandal, "Modeling and design of CMOS analog circuits through hierarchical abstraction," *Integration*, vol. 46, no. 4, pp. 449–462, Sep. 2013.
215. "Introduction to the ginac framework for symbolic computation within the c++ programming language," 2002.
216. N. Bombieri, G. D. Guglielmo, L. D. Guglielmo, M. Ferrari, F. Fummi, G. Pravadelli, F. Stefanni, and A. Venturelli, "HIFSuite: Tools for HDL code conversion and manipulation," in *2010 IEEE International High Level Design Validation and Test Workshop (HLDVT)*. IEEE, Jun. 2010. [Online]. Available: https://doi.org/10.1109/hldvt.2010.5496665
217. Z. Biolek, D. Biolek, and B. V, "Spice model of memristor with nonlinear dopant drift," *Radioengineering*, vol. 18, 06 2009.
218. M. Monton, J. Engblom, and M. Burton, "Checkpointing for virtual platforms and systemc-tlm," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 1, pp. 133–141, 2013.
219. T. Bradde, S. Grivet-Talocia, P. Toledo, A. V. Proskurnikov, A. Zanco, G. C. Calafiore, and P. Crovetti, "Fast simulation of analog circuit blocks under nonstationary operating conditions," *IEEE Transactions on Components, Packaging and Manufacturing Technology*, vol. 11, no. 9, pp. 1355–1368, 2021.
220. A. Tarraf and L. Hedrich, "Behavioral modeling of transistor-level circuits using automatic abstraction to hybrid automata," in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2019, pp. 1451–1456.
221. D. De Jonghe and G. Gielen, "Characterization of analog circuits using transfer function trajectories," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 59, no. 8, pp. 1796–1804, 2012.

222. A. Chinea and S. Grivet-Talocia, "A parallel vector fitting implementation for fast macromodeling of highly complex interconnects," in *19th Topical Meeting on Electrical Performance of Electronic Packaging and Systems*, Oct 2010, pp. 101–104.
223. B. Gustavsen, "Improving the Pole Relocating Properties of Vector Fitting," *IEEE Transactions on Power Delivery*, vol. 21, no. 3, pp. 1587–1592, jul 2006.
224. ——, "Rational modeling of multiport systems via a symmetry and passivity preserving mode-revealing transformation," *IEEE Transactions on Power Delivery*, vol. 29, no. 1, pp. 199–206, 2014.
225. T. Bobach and G. Umlauf, "Natural neighbor interpolation and order of continuity," *GI Lecture Notes in Informatics: Visualization of Large and Unstructured Data Sets*, pp. 69–86, 2006.
226. B.-Q. Mu and H.-F. Chen, "Recursive identification of mimo wiener systems," *IEEE Transactions on Automatic Control*, vol. 58, no. 3, pp. 802–808, 2013.
227. P. F. Cunha and P. G. Maropoulos, *Digital Enterprise Technology - Perspectives and Future Challenges*. Springer, 2007.
228. J. Morgan, M. Halton, Y. Qiao, and J. G. Breslin, "Industry 4.0 smart reconfigurable manufacturing machines," *Journal of Manufacturing Systems*, vol. 59, pp. 481–506, 2021.
229. G. Tristo, G. Bissacco, A. Lebar, and J. Valentinčič, "Real time power consumption monitoring for energy efficiency analysis in micro EDM milling," *International Journal of Advanced Manufacturing Technology*, vol. 78, p. 1511–1521, 2015.
230. R. Stetter, "Approaches for modelling the physical behavior of technical systems on the example of wind turbines," *Energies*, vol. 13, no. 8, 2020.
231. C. Nigischer, S. Bougain, R. Riegler, H. P. Stanek, and M. Grafinger, "Multi-domain simulation utilizing sysml: state of the art and future perspectives," *Procedia CIRP*, vol. 100, pp. 319–324, 2021, 31st CIRP Design Conference 2021 (CIRP Design 2021).
232. R. Leveugle and A. Ammari, "Early seu fault injection in digital, analog and mixed signal circuits: a global flow," in *Proceedings Design, Automation and Test in Europe Conference and Exhibition*, vol. 1, 2004, pp. 590–595 Vol.1.
233. S. Wolny, A. Mazak, C. Carpella, V. Geist, and M. Wimmer, "Thirteen years of SysML: a systematic mapping study," *Software and Systems Modeling*, vol. 19, no. 1, pp. 111–169, May 2019.
234. A. M. Madni, C. C. Madni, and S. D. Lucero, "Leveraging digital twin technology in model-based systems engineering," *Systems*, vol. 7, no. 1, 2019.
235. P. Munk and A. Nordmann, "Model-based safety assessment with sysml and component fault trees: Application and lessons learned," *Softw. Syst. Model.*, vol. 19, no. 4, p. 889–910, jul 2020.
236. R. Barbau, C. Bock, and M. Dadfarnia, "Translator from extended SysML to physical interaction and signal flow simulation platforms," *Journal of Research of the National Institute of Standards and Technology*, vol. 124, Jul. 2019.
237. A. Junghanns, C. Gomes, C. Schulze, K. Schuch, P. R., M. Blaesken, I. Zacharias, A. Pillekeit, K. Wernersson, T. Sommer, C. Bertsch, T. Blochwitz, and M. Najafi, "The functional mock-up interface 3.0 - new features enabling new applications," in *Linköping Electronic Conference Proceedings*. Linköping University Electronic Press, Sep. 2021.
238. J. Parri, F. Patara, S. Sampietro, and E. Vicario, "A framework for model-driven engineering of resilient software-controlled systems," *Computing*, vol. 103, no. 4, p. 589–612, apr 2021.
239. M. Rewieński, "A perspective on fast-SPICE simulation technology," in *Simulation and Verification of Electronic and Biological Systems*. Springer Netherlands, 2011, pp. 23–42.
240. M. Zwolinski and A. D. Brown, "Behavioural modelling of analogue faults in vhdl-ams - a case study," in *2004 IEEE International Symposium on Circuits and Systems (IEEE Cat. No.04CH37512)*, vol. 5, May 2004, pp. V–632–V–635 Vol.5.
241. J. Arlat, Y. Crouzet, J. Karlsson, P. Folkesson, E. Fuchs, and G. H. Leber, "Comparison of physical and software-implemented fault injection techniques," *IEEE Transactions on Computers*, vol. 52, no. 9, pp. 1115–1133, Sep. 2003.

242. D. Gil, J. Gracia, J. Baraza, and P. Gil, “Study, comparison and application of different vhdl-based fault injection techniques for the experimental validation of a fault-tolerant system,” *Microelectronics Journal*, vol. 34, no. 1, pp. 41 – 51, 2003.
243. R. Mariani and G. Boschi, “A systematic approach for failure modes and effects analysis of system-on-chips,” in *13th IEEE International On-Line Testing Symposium (IOLTS 2007)*. IEEE, 2007, pp. 187–188.
244. A. Baklouti, N. Nguyen, F. Mhenni, J.-Y. Choley, and A. Mlika, “Improved safety analysis integration in a systems engineering approach,” *Applied Sciences*, vol. 9, no. 6, p. 1246, 2019.
245. A. Korsunovs, A. Doikin, F. Campean, S. Kabir, E. Hernandez, D. Taggart, S. Parker, and G. Mills, “Towards a model-based systems engineering approach for robotic manufacturing process modelling with automatic fmea generation,” *Proceedings of the Design Society*, vol. 2, pp. 1905–1914, 2022.
246. T. Markwirth, R. Jancke, and C. Sohrmann, “Dynamic fault injection into digital twins of safety-critical systems,” in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021, pp. 446–450.
247. D. G. Pivoto, L. F. de Almeida, R. da Rosa Righi, J. J. Rodrigues, A. B. Lugli, and A. M. Alberti, “Cyber-physical systems architectures for industrial internet of things applications in industry 4.0: A literature review,” *Journal of Manufacturing Systems*, vol. 58, pp. 176–192, 2021.
248. Z. Kang, C. Catal, and B. Tekinerdogan, “Machine learning applications in production lines: A systematic literature review,” *Computers & Industrial Engineering*, vol. 149, p. 106773, 2020.
249. M. Mestre, “Avoiding top pitfalls in annotation projects,” https://towardsdatascience.com, 2021.
250. S. Enderes, “The impact of annotation errors on neural networks,” https://understand.ai, 2021.
251. S. Ahmad, A. Lavin, S. Purdy, and Z. Agha, “Unsupervised real-time anomaly detection for streaming data,” *Neurocomputing*, vol. 262, pp. 134–147, 2017, online Real-Time Learning Strategies for Data Streams.
252. P. Dobbelaere and K. S. Esmaili, “Kafka versus RabbitMQ: A Comparative Study of Two Industry Reference Publish/Subscribe Implementations: Industry Paper,” in *Proc. of ACM DEBS*, 2017, p. 227–238.
253. A. Tsanousa, E. Bektsis, C. Kyriakopoulos, A. G. González, U. Leturiondo, I. Gialampoukidis, A. Karakostas, S. Vrochidis, and I. Kompatsiaris, “A review of multisensor data fusion solutions in smart manufacturing: Systems and trends,” *Sensors*, vol. 22, no. 5, 2022.
254. Özgür Gültekin, E. Cinar, K. Özkan, and A. Yazıcı, “Multisensory data fusion-based deep learning approach for fault diagnosis of an industrial autonomous transfer vehicle,” *Expert Systems with Applications*, vol. 200, p. 117055, 2022.
255. Y. Wei, D. Wu, and J. Terpenny, “Decision-level data fusion in quality control and predictive maintenance,” *IEEE Transactions on Automation Science and Engineering*, vol. 18, no. 1, pp. 184–194, Jan. 2021.
256. K. Imoto, T. Nakai, T. Ike, K. Haruki, and Y. Sato, “A cnn-based transfer learning method for defect classification in semiconductor manufacturing,” *IEEE Transactions on Semiconductor Manufacturing*, vol. 32, no. 4, pp. 455–459, 2019.
257. J. Yu, Z. Shen, and X. Zheng, “Joint feature and label adversarial network for wafer map defect recognition,” *IEEE Transactions on Automation Science and Engineering*, vol. 18, no. 3, pp. 1341–1353, 2021.
258. Y. Roh, G. Heo, and S. E. Whang, “A survey on data collection for machine learning: A big data - ai integration perspective,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 33, no. 4, pp. 1328–1347, 2021.
259. S. Gaiardelli, S. Spellini, M. Panato, M. Lora, and F. Fummi, “A software architecture to control service-oriented manufacturing systems,” in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2022, pp. 40–43.
260. M. Barandas, D. Folgado, L. Fernandes, S. Santos, M. Abreu, P. Bota, H. Liu, T. Schultz, and H. Gamboa, “Tsfel: Time series feature extraction library,” *SoftwareX*, vol. 11, p. 100456, 2020.
261. H. Peng, F. Long, and C. Ding, “Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy,” *IEEE Transactions on pattern analysis and machine intelligence*, vol. 27, no. 8, pp. 1226–1238, 2005.
262. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

263. F. Chollet, “Keras,” https://keras.io, 2015.
264. J. Bergstra and Y. Bengio, “Random search for hyper-parameter optimization.” *Journal of machine learning research*, vol. 13, no. 2, 2012.
265. I. Bozcan, C. Korndorfer, M. W. Madsen, and E. Kayacan, “Score-based anomaly detection for smart manufacturing systems,” *IEEE/ASME Transactions on Mechatronics*, 2022.
266. L. Jing and Y. Tian, “Self-supervised visual feature learning with deep neural networks: A survey,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 43, no. 11, pp. 4037–4058, 2020.
267. Y. Gal and Z. Ghahramani, “Dropout as a bayesian approximation: Representing model uncertainty in deep learning,” in *international conference on machine learning*. PMLR, 2016, pp. 1050–1059.
268. D. Hendrycks, K. Zhao, S. Basart, J. Steinhardt, and D. Song, “Natural adversarial examples,” *CVPR*, 2021.
269. R. Rosen, G. von Wichert, G. Lo, and K. D. Bettenhausen, “About the importance of autonomy and digital twins for the future of manufacturing,” *IFAC Symposium on Information Control Problems in Manufacturing*, vol. 48, no. 3, pp. 567–572, 2015.
270. F. Tao, Q. Qi, A. Liu, and A. Kusiak, “Data-driven smart manufacturing,” *Journal of Manufacturing Systems*, vol. 48, pp. 157–169, Jul. 2018.
271. Y. Zhang, G. Zhang, J. Wang, S. Sun, S. Si, and T. Yang, “Real-time information capturing and integration framework of the internet of manufacturing things,” *International Journal of Computer Integrated Manufacturing*, vol. 28, p. 811–22, 2015.
272. “OPC FOUNDATION: Opc-ua specifications,” [online] Available: https://opcfoundation.org/developer-tools/specifications-unified-architecture, November 2017.
273. D. Agrawal, A. E. Abbadi, S. Antony, and S. Das, “Data management challenges in cloud computing infrastructures,” in *International workshop on databases in networked information systems*, 2010, p. 1–10.
274. W. Ji and L. Wang, “Big data analytics based fault prediction for shop floor scheduling,” *Journal of Manufacturing Systems*, vol. 43, pp. 187–194, 2017.
275. E. Begoli and J. Horey, “Design principles for effective knowledge discovery from big data,” in *Software Architecture and European Conference on Software Architecture*, 2012, p. 215–8.
276. L. Zhou, J. Li, F. Li, Q. Meng, J. Li, and X. Xu, “Energy consumption model and energy efficiency of machine tools: a comprehensive literature review,” *Journal of Cleaner Production*, vol. 112, pp. 3721–3734, 2016.
277. G. Tristo, G. Bissacco, A. Lebar, and J. Valentincic, “Real time power consumption monitoring for energy efficiency analysis in micro EDM milling,” *International Journal of Advanced Manufacturing Technology*, vol. 78, p. 1511–1521, 2015.
278. G. May, I. Barletta, B. Stahl, and M. Taisch, “Energy management in production: A novel method to develop key performance indicators for improving energy efficiency,” *Applied Energy*, vol. 149, pp. 46–61, 2015.
279. F. Liu, J. Xie, and S. Liu, “A method for predicting the energy consumption of the main driving system of a machine tool in a machining process,” *Journal of Cleaner Production*, vol. 105, pp. 171–177, 2015.
280. S. Emec, J. Kruger, and G. Seliger, “Online fault-monitoring in machine tools based on energy consumption analysis and non-invasive data acquisition for improved resource-efficiency,” *Procedia CIRP*, vol. 40, pp. 236–243, 2016.
281. J. Zou, Q. Chang, J. Arinez, and G. Xiao, “Data-driven modeling and real-time distributed control for energy efficient manufacturing systems,” *Energy*, vol. 127, pp. 247–257, 2017.
282. S. Jia, R. Tang, J. Lv, Q. Yuan, and T. Peng, “Energy consumption modeling of machining transient states based on finite state machine,” *International Journal of Advanced Manufacturing Technology*, vol. 88, p. 2305–2320, 2017.
283. K. Alamin, S. Vinco, M. Poncino, N. Dall’Ora, E. Fraccaroli, and D. Quaglia, “Digital twin extension with extra-functional properties,” in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2021, pp. 434–439.
284. I. Ullah, R. Ahmad, and D. Kim, “A prediction mechanism of energy consumption in residential buildings using hidden markov model,” *Energies*, vol. 11, no. 2, 2018.

285. N. Bhusal, R. M. Shukla, M. Gautam, M. Benidris, and S. Sengupta, “Deep ensemble learning-based approach to real-time power system state estimation,” *International Journal of Electrical Power & Energy Systems*, vol. 129, p. 106806, 2021.
286. Şükrü Özşahin and H. Singer, “Development of an artificial neural network model to minimize power consumption in the milling of heat-treated and untreated wood,” Kastamonu University, Tech. Rep., 2019.
287. A. Bakurova, O. Yuskiv, D. S. A. Riabenko, and E. Tereschenko, “Neural network forecasting of energy consumption of a metallurgical enterprise,” *Innovative Technologies and Scientific Solutions for Industries*, vol. 1, 2021.
288. P. Rajan and A. Sekar, “Linear circuit analysis,” in *The Electrical Engineering Handbook*. Academic Press, 2005, pp. 3–29.
289. R. Drath and A. Horch, “Industrie 4.0: Hit or hype?” *IEEE Industrial Electronics Magazine*, vol. 8, no. 2, pp. 56–58, jun 2014.
290. J. Vachalek *et al.*, “The digital twin of an industrial production line within the industry 4.0 concept,” in *2017 21st International Conference on Process Control (PC)*. IEEE, jun 2017.
291. E. A. Lee, “Cyber physical systems: Design challenges,” in *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*. IEEE, may 2008, pp. 363–369. [Online]. Available: https://doi.org/10.1109/isorc.2008.25
292. *IEC 62264 – Enterprise-control system integration*, IEC, 2013. [Online]. Available: https://www.iso.org/standard/57308.html
293. D. Goyal and B. Pabla, “Condition based maintenance of machine tools—a review,” *CIRP Journal of Manufacturing Science and Technology*, vol. 10, pp. 24–35, Aug. 2015.
294. R. K. M. President and C. of Integrated Systems Inc., *An Introduction to Predictive Maintenance (Plant Engineering)*. Butterworth-Heinemann, 2002.
295. H. M. Hashemian, “State-of-the-art predictive maintenance techniques,” *IEEE Transactions on Instrumentation and Measurement*, vol. 60, no. 1, pp. 226–236, jan 2011.
296. F. Nowlan and H. Heap, *Reliability-centered Maintenance*. Dolby Access Press, 1978.
297. Z. Peng and N. Kessissoglou, “An integrated approach to fault diagnosis of machinery using wear debris and vibration analysis,” *Wear*, vol. 255, no. 7-12, pp. 1221–1232, Aug. 2003.
298. M. F. Yaqub, I. Gondal, and J. Kamruzzaman, “Machine fault severity estimation based on adaptive wavelet nodes selection and SVM,” in *2011 IEEE International Conference on Mechatronics and Automation*. IEEE, aug 2011.
299. A. D. Nembhard, J. K. Sinha, A. J. Pinkerton, and K. Elbhbah, “Combined vibration and thermal analysis for the condition monitoring of rotating machinery,” *Structural Health Monitoring*, vol. 13, no. 3, pp. 281–295, 2014.
300. G. S. Martinez, T. Karhela, R. Ruusu, T. Lackman, and V. Vyatkin, “Towards a systematic path for dynamic simulation to plant operation: OPC UA-enabled model adaptation method for tracking simulation,” in *IECON 2017 - 43rd Annual Conference of the IEEE Industrial Electronics Society*. IEEE, Oct. 2017.
301. D. Catenazzo, B. OrFlynn, and M. Walsh, “On the use of wireless sensor networks in preventative maintenance for industry 4.0,” in *2018 12th International Conference on Sensing Technology (ICST)*. IEEE, Dec. 2018.
302. ISO, *ISO 10816:2014 Mechanical vibration – Evaluation of machine vibration by measurements on non-rotating parts*. pub-ISO, 2014, available in electronic form for online purchase at http://www.iso.org/standard/56782.

A

Verilog-A fault templates

Listing A.1: Templates for MOSFET fault models (part 1).

```
1  // DEFINING TEMPLATES FOR MOSFET FAULT MODELS
2  // The present file defines "templates" which are using pre-processor
3  // directives as parameters in order to generate wrappers for multiple
4  // MOSFET models.
5
6  // Macro for the underlying model
7  `ifndef ORIGINAL_MOSFET_MODEL
8  `define ORIGINAL_MOSFET_MODEL `ORIGINAL_MOSFET_MODEL
9  `endif
10
11 // Macro for the wrapper name
12 `ifndef TARGET_MOSFET_MODEL
13 `define TARGET_MOSFET_MODEL anabasis_`ORIGINAL_MOSFET_MODEL
14 `endif
15
16 // ANABASIS_MOSFET Wrapper Module
17 // ++++++++++++++++++++++++++++++
18 //
19 // These modules are fault injection wrappers for MOSFET components.
20
21 module `TARGET_MOSFET_MODEL (D, G, S, B);
22
23     // Interface nodes
24     inout D, G, S, B ;
25     electrical D, G, S, B ;
26
27     // Geometrical instance parameters
28     parameter real l   = 0.35E-6 ;
29     parameter real w   = 10.0E-6 ;
30     parameter real ad  = 0.0 ;
31     parameter real as  = 0.0 ;
32     parameter real pd  = 0.0 ;
33     parameter real ps  = 0.0 ;
34     parameter real nrd = 0.0 ;
35     parameter real nrs = 0.0 ;
36
37     // Failure mode parameters
38     parameter integer failmode = 0 ;
39     parameter real ropen = 1.0E9 ;
40     parameter real rshort = 0.1 ;
41
42     genvar this_mode;
43
44     for (this_mode = failmode; this_mode <= failmode; this_mode = this_mode + 1) begin
45         case (this_mode)
46             (0): begin
47                 // FAULT-FREE
48                 // ----------
49                 // Instanciating the fault-free instance
50                 `ORIGINAL_MOSFET_MODEL # (.l(l),.w(w),.ad(ad),.as(as),.pd(pd),.ps(ps),.nrd(nrd),.nrs(nrs)) core(D, G
     , S, B) ;
51             end
52
53             (1): begin
54                 // DRAIN OPEN
55                 // ----------
56                 // Internal nodes
57                 electrical DD;
58
59                 // Instanciating the fault-free instance
60                 `ORIGINAL_MOSFET_MODEL # (.l(l),.w(w),.ad(ad),.as(as),.pd(pd),.ps(ps),.nrd(nrd),.nrs(nrs)) core(DD,
     G, S, B) ;
61
62                 // Drain open
63                 resistor #(.r(ropen)) defect (D, DD) ;
64             end
```

```
1             (2): begin
2                 // GATE OPEN
3                 // ---------
4                 // Internal nodes
5                 electrical GG;
6 
7                 // Instanciating the fault-free instance
8                 `ORIGINAL_MOSFET_MODEL # (.l(l),.w(w),.ad(ad),.as(as),.pd(pd),.ps(ps),.nrd(nrd),.nrs(nrs)) core(D,
       GG, S, B) ;
9 
10                // Gate open
11                resistor #(.r(ropen)) defect (G, GG) ;
12            end
13 
14            (3): begin
15                // SOURCE OPEN
16                // -----------
17                // Internal nodes
18                electrical SS;
19 
20                // Instanciating the fault-free instance
21                `ORIGINAL_MOSFET_MODEL # (.l(l),.w(w),.ad(ad),.as(as),.pd(pd),.ps(ps),.nrd(nrd),.nrs(nrs)) core(D, G
       , SS, B) ;
22 
23                // Gate open
24                resistor #(.r(ropen)) defect (S, SS) ;
25            end
26 
27            (4): begin
28                // BULK OPEN
29                // ----------
30                // Internal nodes
31                electrical BB;
32 
33                // Instanciating the fault-free instance
34                `ORIGINAL_MOSFET_MODEL # (.l(l),.w(w),.ad(ad),.as(as),.pd(pd),.ps(ps),.nrd(nrd),.nrs(nrs)) core(D, G
       , S, BB) ;
35 
36                // Gate open
37                resistor #(.r(ropen)) defect (B, BB) ;
38            end
39 
40            (5): begin
41                // GATE-DRAIN SHORT
42                // ----------------
43                // Instanciating the fault-free instance
44                `ORIGINAL_MOSFET_MODEL # (.l(l),.w(w),.ad(ad),.as(as),.pd(pd),.ps(ps),.nrd(nrd),.nrs(nrs)) core(D, G
       , S, B) ;
45 
46                // Gate-Drain short
47                resistor #(.r(rshort)) defect (G, D) ;
48            end
49 
50            (6): begin
51                // GATE-SOURCE SHORT
52                // ----------------
53                // Instanciating the fault-free instance
54                `ORIGINAL_MOSFET_MODEL # (.l(l),.w(w),.ad(ad),.as(as),.pd(pd),.ps(ps),.nrd(nrd),.nrs(nrs)) core(D, G
       , S, B) ;
55 
56                // Gate-Source short
57                resistor #(.r(rshort)) defect (G, S) ;
58            end
59 
60            (7): begin
61                // GATE-BODY SHORT
62                // ----------------
63                // Instanciating the fault-free instance
64                `ORIGINAL_MOSFET_MODEL # (.l(l),.w(w),.ad(ad),.as(as),.pd(pd),.ps(ps),.nrd(nrd),.nrs(nrs)) core(D, G
       , S, B) ;
65 
66                // Gate-Source short
67                resistor #(.r(rshort)) defect (G, B) ;
68            end
69 
70            (8): begin
71                // BODY-DRAIN SHORT
72                // ----------------
73                // Instanciating the fault-free instance
74                `ORIGINAL_MOSFET_MODEL # (.l(l),.w(w),.ad(ad),.as(as),.pd(pd),.ps(ps),.nrd(nrd),.nrs(nrs)) core(D, G
       , S, B) ;
75 
76                // Body-Drain short
77                resistor #(.r(rshort)) defect (B, D) ;
78            end
79 endmodule
```

Listing A.2: Templates for MOSFET fault models (part 2).

```
1               (9): begin
2                   // BODY-SOURCE SHORT
3                   // ----------------
4                   // Instanciating the fault-free instance
5                   `ORIGINAL_MOSFET_MODEL # (.l(l),.w(w),.ad(ad),.as(as),.pd(pd),.ps(ps),.nrd(nrd),.nrs(nrs)) core(D, G
      , S, B) ;
6
7                   // Body-Drain short
8                   resistor #(.r(rshort)) defect (B, S) ;
9               end
10          (10): begin
11                  // DRAIN-SOURCE SHORT
12                  // ----------------
13                  // Instanciating the fault-free instance
14                  `ORIGINAL_MOSFET_MODEL # (.l(l),.w(w),.ad(ad),.as(as),.pd(pd),.ps(ps),.nrd(nrd),.nrs(nrs)) core(D, G
      , S, B) ;
15
16                  // Body-Drain short
17                  resistor #(.r(rshort)) defect (D, S) ;
18              end
19
20              // Absolutely necessary in order to handle compilation when instances rely on the default value of the
      failmode parameter.
21              default: begin
22                  // FAULT-FREE (as DEFAULT)
23                  // -----------------------
24                  // Instanciating the fault-free instance
25                  `ORIGINAL_MOSFET_MODEL # (.l(l),.w(w),.ad(ad),.as(as),.pd(pd),.ps(ps),.nrd(nrd),.nrs(nrs)) core(D, G
      , S, B) ;
26              end
27
28          endcase
29  endmodule
```

Listing A.3: Templates for MOSFET fault models (part 3).

B

Verilog-AMS faulty circuits

```
1  `include "disciplines.vams"
2  `include "constants.vams"
3  `timescale 1us / 1us
4
5  module rlc (t1, t4);
6      // PARAMETERS ----------------------
7      parameter real r = 7;
8      parameter real l = 0.15;
9      parameter real c = 100e-6;
10     // PORTS -------------------------
11     input t1, t4;
12     // NODES -------------------------
13     electrical t1, t2, t3, t4, g;
14     // BRANCHES ----------------------
15     branch(t1,g) res;
16     branch(t2,t3) ind;
17     branch(t3,t4) cap;
18     branch(g,t2) fault;
19     // BEHAVIOR ----------------------
20     analog begin
21         I(res) <+ V(res)/r;
22         I(ind) <+ idt(V(ind))/l;
23         I(cap) <+ ddt(V(cap))*c;
24         I(fault) <+ V(fault) / 1e09;
25     end
26 endmodule
```

Listing B.1: Faulty RLC circuit modeled in Verilog-AMS, open fault.

```
1  `include "disciplines.vams"
2  `include "constants.vams"
3  `timescale 1us / 1us
4
5  module rlc (t1, t4);
6      // PARAMETERS ----------------------
7      parameter real r = 7;
8      parameter real l = 0.15;
9      parameter real c = 100e-6;
10     // PORTS ---------------------------
11     input t1, t4;
12     // NODES ---------------------------
13     electrical t1, t2, t3, t4;
14     // BRANCHES ------------------------
15     branch(t1,t2) res;
16     branch(t2,t3) ind;
17     branch(t3,t4) cap;
18     branch(t1,t2) fault;
19     // BEHAVIOR ------------------------
20     analog begin
21         I(res) <+ V(res)/r;
22         I(ind) <+ idt(V(ind))/l;
23         I(cap) <+ ddt(V(cap))*c;
24         I(fault) <+ V(fault) / 1;
25     end
26 endmodule
```

Listing B.2: Faulty RLC circuit modeled in Verilog-AMS, short fault.

```
1  `include "disciplines.vams"
2  `include "constants.vams"
3  `timescale 1us / 1us
4
5  module double_rlc (t1, t6);
6      // PARAMETERS ----------------------
7      parameter real r0 = 7;
8      parameter real l0 = 0.15;
9      parameter real c0 = 100e-6;
10     parameter real r1 = 5;
11     parameter real l1 = 0.10;
12     parameter real c1 = 50e-6;
13     // PORTS ---------------------------
14     inout t1, t6;
15     // NODES ---------------------------
16     electrical t1, t2, t3 , t4, t5, t6, g;
17     // BRANCHES ------------------------
18     branch(t1,t2) res0;
19     branch(t2,t3) ind0;
20     branch(t3,t4) cap0;
21     branch(t4,t5) cap1;
22     branch(t5,g) res1;
23     branch(t4,t6) ind1;
24     branch(g, t6) fault;
25     // BEHAVIOR ------------------------
26     analog begin
27
28         V(res0) <+ I(res0) * r0;
29         V(ind0) <+ l0*ddt(I(ind0));
30         V(cap0) <+ idt(I(cap0))/c0;
31         V(cap1) <+ idt(I(cap1))/c1;
32         V(res1) <+ I(res1) * r1;
33         V(ind1) <+ l1*ddt(I(ind1));
34         V(fault) <+ I(fault) * 1e09;
35     end
36 endmodule
```

Listing B.3: Faulty Double-RLC circuit modeled in Verilog-AMS, open fault.

```
1  `include "disciplines.vams"
2  `include "constants.vams"
3  `timescale 1us / 1us
4
5  module double_rlc (t1, t6);
6      // PARAMETERS -----------------------
7      parameter real r0 = 7;
8      parameter real l0 = 0.15;
9      parameter real c0 = 100e-6;
10     parameter real r1 = 5;
11     parameter real l1 = 0.10;
12     parameter real c1 = 50e-6;
13     // PORTS ----------------------------
14     inout t1, t6;
15     // NODES ----------------------------
16     electrical t1, t2, t3 , t4, t5, t6;
17     // BRANCHES -------------------------
18     branch(t1,t2) res0;
19     branch(t2,t3) ind0;
20     branch(t3,t4) cap0;
21     branch(t4,t5) cap1;
22     branch(t5,t6) res1;
23     branch(t4,t6) ind1;
24     branch(t5, t6) fault;
25     // BEHAVIOR -------------------------
26     analog begin
27         V(res0) <+ I(res0) * r0;
28         V(ind0) <+ l0*ddt(I(ind0));
29         V(cap0) <+ idt(I(cap0))/c0;
30         V(cap1) <+ idt(I(cap1))/c1;
31         V(res1) <+ I(res1) * r1;
32         V(ind1) <+ l1*ddt(I(ind1));
33         V(fault) <+ I(fault) * 1;
34     end
35 endmodule
```

Listing B.4: Faulty Double-RLC circuit modeled in Verilog-AMS, short fault.