

High-Level Analysis of the Impact of Soft-Faults in Cyberphysical Systems

Marwan Ammar

A thesis
in The Department
of
Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of
Doctor of Philosophy (Electrical and Computer Engineering) at
Concordia University
Montréal, Québec, Canada

November 2019

© Marwan Ammar, 2019

CONCORDIA UNIVERSITY
SCHOOL OF GRADUATE STUDIES

This is to certify that the thesis prepared

By: Marwan Ammar

Entitled: High-Level Analysis of the Impact of Soft-Faults in Cyberphysical Systems

and submitted in partial fulfillment of the requirements for the degree of

Doctor Of Philosophy (Electrical and Computer Engineering)

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

_____ Chair
Dr. Luis Amador-Jimenez

_____ External Examiner
Dr. Roni Khazaka

_____ External to Program
Dr. Lingyu Wang

_____ Examiner
Dr. Abdelwahab Hamou-Lhadj

_____ Examiner
Dr. Ferhat Khendek

_____ Thesis Co-Supervisor
Dr. Otmane Ait Mohamed

_____ Thesis Co-Supervisor
Dr. Yvon Savaria

Approved by _____
Dr. Rastko Selmic, Graduate Program Director

January 8, 2020

Dr. Amir Asif, Dean
Gina Cody School of Engineering & Computer Science

Abstract

High-Level Analysis of the Impact of Soft-Faults in Cyberphysical Systems

Marwan Ammar, Ph.D.

As digital systems grow in complexity and are used in a broader variety of safety-critical applications, there is an ever-increasing demand for assessing the dependability and safety of such systems, especially when subjected to hazardous environments. As a result, it is important to identify and correct any functional abnormalities and component faults as early as possible in order to minimize performance degradation and to avoid potential perilous situations. Existing techniques often lack the capacity to perform a comprehensive and exhaustive analysis on complex redundant architectures, leading to less than optimal risk evaluation. Hence, an early analysis of dependability of such safety-critical applications enables designers to develop systems that meets high dependability requirements. Existing techniques in the field often lack the capacity to perform full system analyses due to state-explosion limitations (such as transistor and gate-level analyses), or due to the time and monetary costs attached to them (such as simulation, emulation, and physical testing).

In this work we develop a system-level methodology to model and analyze the effects of Single Event Upsets (SEUs) in cyberphysical system designs. The proposed methodology investigates the impacts of SEUs in the entire system model (fault tree level), including SEU propagation paths, logical masking of errors, vulnerability to specific events, and critical nodes. The methodology also provides insights on a system's weaknesses, such as the impact of each component to the system's vulnerability, as well as hidden sources of failure, such as latent faults. Moreover, the proposed methodology is able to identify and categorize the system's components in order of criticality, and to evaluate different approaches to the mitigation of such criticality (in the form of different configurations of TMR) in order to obtain the most efficient mitigation solution available.

The proposed methodology is also able to model and analyze system components individually (system component level), in order to more accurately estimate the component's vulnerability to SEUs. In this case, a more refined analysis of the component is conducted, which enables us to identify the source of the component's criticality. Thereafter, a second mitigation mechanic (internal to the component) takes place, in order to evaluate the gains and costs of applying different configurations of TMR to the component internally. Finally, our approach will draw a comparison between the results obtained at both levels of analysis in order to evaluate the most efficient way of improving the targeted system design.

Acknowledgments

It has been an amazing experience and a privilege to pursue my doctorate in the Hardware Verification Group (HVG) at Concordia. It certainly would not have happened without the support and guidance of several people to whom I owe a great deal.

First of all, I would like to thank my supervisor, Dr. Otmane Ait Mohamed. It is he who offered me the opportunity to join the group. He was fully supportive, understanding, involved and present during all the phases of my research. I have learned many things from him in regard to research, academia, and life in general.

Secondly, I sincerely thank my co-supervisor, Prof. Yvon Savaria, for always being there to listen and to offer me guidance in the moments I needed the most. This thesis would not have been possible without his advice, his support and his encouragement.

Next, I'd like to thank all the members of HVG for their help and encouragement, particularly to Dr. Ghaith Bany Hamad.

Last but not least, I thank my close family, my parents, and my little bunny. They have a great deal of credit on everything I have accomplished in my life and for everything I'm yet to accomplish in the future.

Contributions of Authors

Article I: *Formal Analysis of Fault Tree using Probabilistic Model Checking: A Solar Array Case Study*

This paper was the result of my first effort into modeling and analyzing safety-critical systems exposed to single-event upsets (SEUs). The idea was to apply probabilistic model checking to resolve the system's fault diagram expressed in fault tree notation, which is normally resolved through simulation. In this article, I am responsible for all the methodology reasoning, all the modeling, all the experiments and interpretation of the data. The contribution of the co-author (Khaza Anuarul Hoque) were helping to find the case-study and reviewing the paper before submission.

Article II: *Efficient Probabilistic Fault Tree Analysis of Safety Critical Systems via Probabilistic Model Checking*

In this article, I have conducted a study on the impact of redundant architectures (namely, triple modular redundancy) on systems exposed to SEUs. The methodology proposed in this article is the first of its kind, utilizing probabilistic model-checking and high-level system modeling through fault trees to locate possible system vulnerabilities and evaluate the best mitigation solution. In this article, I am responsible for all the modeling, methodologies applied, experiments and data interpretation. The co-author in this work (Ghaith Bany Hamad) helped me to conduct model validation and reviewing the text.

Article III: *System-Level Analysis of the Vulnerability of Processors Exposed to Single-Event Upsets via Probabilistic Model Checking*

This article presents my component-level analysis methodology. Based on my previous work, I identified two sources of problems with fault tree analysis: 1) the high-level of abstraction of fault trees lacks the information to solve some problems, such as

“where does the fault originate from” or “how to avoid it”; 2) in the literature, fault trees are only generated after the system is manufactured and running. How can I generate a fault tree early in the design cycle? Initially, this idea was conceived (and subsequently published) in a paper entitled “Comprehensive Vulnerability Analysis of Systems Exposed to SEUs via Probabilistic Model Checking”, published in the 16th IEEE Conference on Radiation Effects on Components and Systems (RADECS) in 2016. For this article, I have conducted the SEU analysis of two microprocessors. The analysis was conducted with probabilistic model-checking and the modeling was done based on the Register-transfer level (RTL) representation of the targeted processors. Furthermore, one of the products of this approach is the generation of a fault-propagation diagram, that can be easily converted into a fault tree. I was responsible for the conception of the idea, all the modeling and experiments and interpretation of results. The co-author (Ghaith Bany Hamad) offered help with the interpretation of the RTL diagrams, model validation and text review. This article was invited for extension and subsequently published as a chapter in the book Radiation Effects on Integrated Circuits and Systems for Space Applications [136].

Article IV: *Reliability Analysis of the SPARC V8 Architecture via Fault Trees and UPPAL-SMC*

This article was my first attempt at solving a new issue that was highlighted to me by the collection of my previous works. This issue was that fault trees were fundamentally not suitable for an early system analysis of SEU exposure. The reason for this is that a traditional fault tree can only capture a single moment in time. However, SEUs are dynamic events that may occur and reoccur at random intervals. Furthermore, the effects of SEUS may be permanent or only temporary, and the only way to capture these phenomena is to perform a dynamic analysis over time. This paper utilizes my methodology to obtain a fault tree from an RTL diagram and it presents my first effort to perform a fault tree analysis over time. I am responsible for the idea, all the modeling and experiments, as well as interpretation of the results. The co-author (Ghaith Bany Hamad) has helped me with the interpretation of the SPARC V8 architecture and by reviewing the text.

Article V: *Towards an Accurate Probabilistic Modeling and Statistical Analysis of*

Temporal Faults via Temporal Dynamic Fault-Trees (TDFTs)

This article presents the culmination of the idea introduced by *Article IV*. My work has demonstrated that the analysis over time of conventional fault trees can only produce accurate results for a very short period or time, after which the results become unreliable (based on comparison with results obtained by radiation ground testing). To address this, I have proposed a completely new type of time-enabled fault trees, with dynamic properties and dynamic events. Along with a new analysis methodology, the conducted experiments and the results demonstrate that this technique can be used early in the design cycle to predict SEU effects in complex systems. In this paper, I am responsible for all the modeling and analysis and all the experiments and data interpretation. The co-author (Ghaith Bany Hamad) has helped me to conduct model validation and has reviewed the text.

Applications: In addition to the articles presented in the body of this thesis, I have been co-author and have applied my methodology in two journal papers, shown in Chapter 10. In the first article, titled “New Insights Into Soft-Faults Induced Cardiac Pacemakers Malfunctions Analyzed at System-Level Via Model Checking” [21], we perform a comprehensive component-level analysis on a common model of implantable pacemaker exposed to SEUs. In this paper, I am responsible for all the modeling and analysis, all the experiments and for writing about 40% of the text. The first author (Ghaith Bany Hamad) is responsible for the idea of the work, all the background research, the model validation and for writing about 60% of the text. Both authors contributed to the interpretation of the results. In the second article, titled “System-Level Characterization, CTMDP Modeling, and Analysis of Computing Systems Reliability Applied to the LEON3 Processor” (not submitted) we work on improving my component-level methodology by proposing a system-level analysis based on the application that the system under test is running. This involves the characterization of the micro architecture, as well as each possible operation and the sequence of operations in the application. Initially, this idea was conceived (and subsequently published) in a paper entitled “System-Level Characterization, Modeling, and Probabilistic Formal Analysis of LEON3 Vulnerability to Transient Faults”, published in the 18th IEEE Conference on Radiation Effects on Components and Systems (RADECS) in 2018. In this article, I am responsible for all the modeling

and analysis, all the experiments and for writing approximately 40% of the text. The first author (Ghaith Bany Hamad) is responsible for the characterization of the LEON3 architecture and for writing approximately 60% of the text. Both authors contributed to the characterization of the applications and the interpretation of the results.

Contents

List of Figures	xv
List of Tables	xviii
1 Introduction	1
1.1 Motivation and Problem Formulation	3
1.1.1 Radiation Ground Testing	4
1.1.2 Radiation Testing through Simulation	5
1.1.3 Radiation Testing through Formal Verification	6
1.2 Thesis Objectives	6
1.3 Thesis Contributions	7
1.3.1 Fault-Tree Analysis Phase	9
1.3.2 Component-Level Analysis Phase	12
1.4 Thesis Organization	15
2 Literature Review	18
3 Background Information	21
3.1 Fault Tree Analysis	21
3.2 Triple Modular Redundancy	23
3.3 Probabilistic Model Checking	24
3.3.1 PRISM Model Checker	24
3.3.2 UPPAAL-SMC	25
4 Article I: Formal Analysis of Fault Tree using Probabilistic Model Checking: A Solar Array Case Study	27
4.1 Introduction	28

4.2	Preliminaries	29
4.2.1	Probabilistic Model Checking with PRISM	29
4.2.2	Fault Tree Analysis	30
4.3	Related Works	31
4.4	Modeling	33
4.4.1	Modeling of an AND Gate	33
4.4.2	Modeling of an OR Gate	35
4.4.3	Sample Modeling of a 2-Gate System	36
4.5	Case Study	38
4.6	Conclusion	41
5	Article II: Efficient Probabilistic Fault Tree Analysis of Safety Critical Systems via Probabilistic Model Checking	43
5.1	Introduction	44
5.2	Related Work	46
5.3	Proposed Methodology	47
5.3.1	Probabilistic Modeling of Fault Trees	48
5.3.2	Modeling of TMR	52
5.4	Experimental Results	54
5.5	Conclusion	60
6	Article III: System-Level Analysis of the Vulnerability of Processors Exposed to Single-Event Upsets via Probabilistic Model Checking	62
6.1	Introduction	63
6.2	Markov Modeling of Self-Repair Systems	65
6.3	Proposed Probabilistic Modeling of SEUs Propagation in Processors .	67
6.3.1	Fetching Phase	69
6.3.2	Decoding Phase	70
6.3.3	Execution Phase	71
6.3.4	Self-Repair Routine	72
6.4	Proposed Formal Modeling and Analysis in PRISM	72
6.5	Experimental Analysis	74
6.6	Conclusion and Future Work	81

7	Article IV: Reliability Analysis of the SPARC V8 Architecture via Fault Trees and UPPAL-SMC	82
7.1	Introduction	82
7.2	Existing Fault Analysis of SPARC V8	84
7.3	Modeling the SPARC V8 Pipeline as DFT	84
7.3.1	General Considerations and Assumptions	85
7.3.2	System Level Fault Abstraction	85
7.3.3	PTA Model Composition	87
7.4	Stochastic Soft-fault Analysis with UPPAAL	88
7.5	Conclusion and Future Work	91
8	Article V: Towards an Accurate Probabilistic Modeling and Statistical Analysis of Temporal Faults via Temporal Dynamic Fault-Trees (TDFTs)	92
8.1	Introduction	93
8.2	Related Works	95
8.3	Preliminaries	97
8.3.1	The UPPAAL Formalism	97
8.3.2	Fault Tree Analysis	97
8.4	Proposed TDFT Modeling and Analysis Methodology	98
8.4.1	Proposed Probabilistic Model of the Temporal AND Gate . .	100
8.4.2	Proposed Probabilistic Model of the Temporal OR Gate . . .	103
8.4.3	Proposed Probabilistic Model of the Temporal FDEP gate . .	104
8.4.4	Proposed Probabilistic Model of the Temporal PAND gate . .	106
8.4.5	Proposed Probabilistic Model of the Temporal COMB Gate .	107
8.4.6	Proposed Analysis Methodology	110
8.5	Experimental Results	111
8.5.1	Unreliability Evaluation Over Time	111
8.5.2	Scalability of the Proposed TDFT Analysis	112
8.5.3	Comparison between TDFTs and Temporal Fault Trees (TFTs)	116
8.5.4	Failure Estimation of the SPARC V8 Architecture with TDFTs	117
8.6	Conclusion	120

9	Conclusion and Future Work	122
9.1	Conclusion	122
9.2	Future Work	124
10	Applications	127
10.1	New Insights Into Soft-Faults Induced Cardiac Pacemakers Malfunc- tions Analyzed at System-Level Via Model Checking	127
10.1.1	Introduction	128
	Existing Observations of the Impact of Soft-Faults on Pacemakers	130
	Formal Modeling and Analysis of the Functionality of the Pace- makers	131
10.1.2	Probabilistic Model Checking (PMC) & <i>Storm</i>	132
10.1.3	Steps of the Proposed Pacemaker Analysis	132
10.1.4	Behavior of the DDD Pacemaker	134
10.1.5	PTA Modeling & Functional Analysis of Pacemaker	135
10.1.6	Non-functional Analysis of the Pacemaker Vulnerability to Soft- Faults	139
	Soft-Fault Classification	140
	Formal Modeling and Analysis of Soft-Faults	141
10.1.7	New Insights on Possible Pacemaker Malfunctions Induced by Soft-Faults	145
	SF-Induced Pacemaker Oversensing	146
	SF-Induced Pacemaker Undersensing	150
	SF-Induced Output Failure	151
10.1.8	Conclusion	151
10.2	System-Level Characterization, CTMDP Modeling, and Analysis of Computing Systems Reliability Applied to the LEON3 Processor . . .	152
10.2.1	Introduction	153
10.2.2	Main Steps of the Proposed Framework	156
10.2.3	Instruction Based Characterization, Modeling, and Analysis .	159
	Characterization of Radiation-Induced Soft-Errors	159
	Microarchitecture-Based Characterization	160
	Instruction-Based Markov Modeling	161
10.2.4	Fault Injection and Analysis Through CTMDP	162

10.2.5 Application Based SEU Modeling and Analysis	165
10.2.6 Experimental Results	166
10.2.7 Conclusion	170
Bibliography	171

List of Figures

1	Overview of the Proposed Framework	9
2	Example of TMR Applied to a Generic Component	23
3	Illustrative Example of the UPPAAL Formalism. Reproduced from [46].	26
4	Fault Tree Gates	33
5	2-input AND gate DTMC	34
6	PRISM modeling of an AND gate	35
7	2 input OR gate DTMC	36
8	PRISM modeling of an OR gate	36
9	2-gate DTMC	37
10	PRISM modeling of the 2-gate example	37
11	Solar Array Fault Tree	38
12	Redundancy Test	41
13	Main Steps of the Proposed Methodology	48
14	Example of Fault Tree Gate Automata	50
15	Modeling FT as MDP versus DTMC	52
16	Example of different TMR arrangements	54
17	Results of Investigating the Proposed Methodology’s Scalability . . .	56
18	TMR chain with uniform failure rates	57
19	Variation in the system failure rate due to non-uniform distribution .	58
20	Hermes Cubesat HSCOM Fault Tree	59
21	Hermes HSCOM Results	59
22	Solar Array TMR Results	60
23	Time Progression of a Fault-Tolerant Micro-Electronic System Exposed to SEUs [9]	66
24	Proposed Probabilistic Model of SEU Propagation Through a Proces- sor Instruction Cycle.	67

25	Instruction Cycle Example	68
26	FSM of the Self-Repair Routine [9]	73
27	Mean Time to Failure	76
28	Mean Time to Recover	76
29	Steady State Availability	78
30	Vulnerability of Different Registers	79
31	DFT of the 7-stage integer pipeline of the SPARC V8	86
32	Sample of an OR Gate PTA	88
33	Probability of Trap Exceptions in different approaches. Simulation and radiation test results are reproduced from [29]	89
34	Illustrative Example of the UPPAAL Formalism. Reproduced from [46].	98
35	Possible Time Window of a TAND output	100
36	Example of a 2-Input Leaf TAND Gate	101
37	Example of a Simple Fault tree	102
38	Example of a 2-Input Variant TAND Gate	102
39	Example of a 2-Input Leaf TOR Gate	104
40	Example of a TFDEP Gate	105
41	Example of a 2-Input Leaf TPAND Gate	106
42	Example of a 2-of-3 Leaf TComb Gate	107
43	Main Steps of the Proposed Methodology	108
44	Comparison of the Estimated Unreliability Over Time of TDFTs and DFTs	109
45	Modular FT Analysis of the Binary Hypercube Architecture	111
46	Temporal Fault Tree of the Pressure Chamber Case-Study	116
47	DFT of the 7-stage Integer Pipeline of the SPARC-V8 Architecture .	118
48	Probability of Trap Exceptions in different approaches. Simulation and radiation test results are reproduced from [29]	119
49	Main Steps of the Proposed Analysis of a Pacemaker	133
50	TAs of the Components of the DDD Pacemaker	134
51	Proposed Formal Analysis of <i>Soft-Faults</i> Propagation	139
52	Effects of SFs on the Pacemaker Components	143
53	Timing Diagram of SF at <i>Aget</i> During TPVAB	147
54	Timing Diagram of an SF at <i>Aget</i> During TPVARP	148

55	Timing Diagram of the Impact of SF During TURI	148
56	Timing Diagram of the Impact of SF During TVRP	149
57	Timing Diagram of SF-Induced Oversensing	149
58	Timing Diagram of SF-Induced Oversensing	150
59	SF-Induced Undersensing	151
60	Output Failure due to Missed Ventricular Pacing	152
61	The main steps of the proposed methodology.	157
62	Example of C++ to LLVM Conversion	158
63	Proposed Probabilistic Model of SEU Propagation Through a the Pro- cessor Pipeline in an ADD Instruction	161
64	Example of Variable Lifetime Estimation	163
65	Probabilistic Model of SEU Propagation Through a Sequence of In- structions Based on the Target Application.	163
66	Criticality Evaluation of the LEON3 Pipeline Registers (CRC Bench- mark). Injection and Chibani 2014 results reproduced from [40] . . .	169

List of Tables

1	TMR configurations and Respective FTs	22
2	Elements of a Fault Tree	31
3	Fault Probability of Bottom Events	40
4	Top Event Failure Probability(DTMC)	40
5	Top Event Failure Probability(MDP)	40
6	Modeled FT gates	51
7	TMR configurations and Respective FTs	53
8	Operations	69
9	MTTR and MTTF in Different Techniques	80
10	AVR ATmega103 Analysis Comparison	80
11	SPARC V8 Probability of Failure Over Time	90
12	Estimated Availability of Component T1 After 100 Seconds. Failure rate of basic events is assumed to be 0.05. Temporal events are assumed to last up to 3 seconds.	113
13	Estimated Availability of Component T2 After 100 Seconds. Failure rate of basic events is assumed to be 0.05. Temporal events are assumed to last up to 3 seconds	113
14	Estimated Availability of the Binary Hypercube System After 100 Seconds. Temporal events are assumed to last up to 3 seconds	114
15	Estimated Reliability of the Pressure Chamber System After 100 Seconds. (Fault=X in the table refers to the time duration of the fault, with X being units of time.)	115
16	SPARC-V8 Probability of Failure Over Time	120
17	Results of the verification of the functional properties of the pacemaker	136
18	Results of the verification of the non-functional properties related to the impact of SFs on the pacemaker	142

19	Application Based Analysis of Fault Propagation	168
20	Average Fault Propagation and Fault Latency.	168
21	Analysis Time for Criticality Evaluation	169

Chapter 1

Introduction

Cyber-physical systems (CPS) are a new class of embedded Information and Communication Technologies (ICT) systems. These systems require tight integration of computing, communication, and control technologies to achieve performance, stability, reliability, efficiency and robustness in physical systems targeting many application domains. Embedded systems have been successfully employed in almost every aspect of our daily lives, ranging from medical devices, buildings, mobile devices, robots, transportation, and energy systems. Such applications impose requirements that are among the most challenging for CPSs being designed today. Furthermore, advancements in technology are likely to increase the complexity of these CPSs even more, with systems required to perform more functions while being smaller in size. Combined with tight time, cost, and design constraints, these factors contribute to making the development of CPSs a major technical challenge. In addition, many CPSs are critical in nature, and they must be highly dependable, even in unknown and hostile environments. This requires powerful methods for failure detection, diagnosis, and recovery to ensure correct system operation. For instance, the failure rate per chip has been reported to increase 100-fold from the 180nm to the 16nm CMOS technology node [111]. Exponential growth in the number of transistors per chip with time has brought tremendous progress in the performance of semiconductor devices; however, it may also increase the device's vulnerability to some types of radiation.

State-of-the-art verification techniques that investigate the unreliability of safety-critical CPSs are very costly, time consuming, and inefficient. In industry, the verification of CPSs is mainly conducted at high level through simulation based techniques,

or at low level through hardware testing. Simulation based techniques are less than optimal, since they rely on the generation of input vectors, which means that the full verification of a complex design is generally unattainable. Moreover, the shortcomings of these techniques are made evident in scenarios where exceptional or low probability events (e.g., failure caused due to non-functional sources) have to be evaluated. For example, despite all the testing and certification that medical devices are submitted to, there are several reports of death induced by pacemaker malfunctions. Reportedly, these malfunctions have originated from the unforeseen effects of external radiation originated in MRI machines, x-rays, and even cosmic ray exposure during a commercial flight [58, 111]. The main challenge in the analysis of CPSs comes from the fact that these systems are very complex, since they comprise several sub-systems and sub-components. The verification of such systems requires the analysis of the vulnerability of all the sub-components and sub-systems individually, as well as the analysis of the interactions between them. Safety-critical systems are often time-critical. Thus, the safety of these systems depends on their ability to correctly collect and process data with real-time requirements. Specification and verification of timing constraints further adds to the complexity of the system. Therefore, efficient ways to analyze these complex systems are of decisive importance. Furthermore, a new methodology which integrates different analysis techniques of this type of systems is required. This methodology must be able to accommodate all the sub-components of the system, and the synergies between them, at different levels of abstraction.

Due to the previously mentioned complexity and composition of cyber-physical systems, the efficient and effective design of distributed multi-scale systems is still an unsolved problem. The design process of these systems must encompass heterogeneous components, often uncertain in specification, their interconnections, and their relationship to the environment. The dynamic of all these elements is critical to the reliability of the system. Technology advances further increase the challenges to the design process, adding the possibility of placing significantly more functionality into products, but also increasing interconnectivity at the risk of unwanted system interactions. Currently in the industry, overdesign is the most used path for safe system design and deployment. However, due to the reasons explained above, this approach is rapidly becoming intractable and it will soon reach its saturation point. Furthermore, some critical CPSs are required to operate with very tight power, area,

and cost constraints. Medical implants, for example, are required to be very small and to have very low power consumption, in order to operate within the patient's body for extended periods of time. To obtain reliable CPSs without excessive overdesign, we need a rigorous methodology for system-level functional verification that is able to: 1) Provide guarantees of performance and reliability against the requirements, even in extremely harsh environments. 2) Produce scalable, fast and cheap verification environments for complex CPS. 3) Exploit analytical tools and techniques to determine design choices and ensure robust system performance. 4) Achieve these goals through the coordinated execution of a prescriptive, repeatable, and measurable process. These points are further elaborated below, in Section 1.3.

1.1 Motivation and Problem Formulation

The failure of a critical CPS can be due to different classes of uncertainties, such as manufacturing defects, aging, end of life failures, and transient faults. An example of the existing reliability analysis of CPSs is the analysis of the impact transient faults due to external radiation have on implantable devices. Radiation is of special interest in critical CPSs as they keep occurring even if these systems are otherwise fault-free and defect-free, in which case such systems could wrongly be expected to have a 0-failure rate. Due to the criticality of many CPSs, the most accepted way of evaluating their vulnerability to the effects of radiation is through a process called dynamic radiation testing [82]. This method consists in exposing the target system to a radiation flux and counting the number of errors observed. However, this method is very expensive and time-consuming, since any change in the application requires a new dynamic test. Alternative methods have emerged, with the goal of reducing the time and cost constraints associated with dynamic radiation ground testing. These techniques are mostly based on system vulnerability analysis through emulation and simulation [17]. However, current techniques are not able to scale to the complexity of these systems. Moreover, such analysis normally consumes large amounts of time and requires full details of the design structure and of the characteristics of the fault. In other words, with detailed circuit level techniques, normally required for detailed modeling, this type of analysis would be intractable at the chip level and is only tractable at the cell level (for hundreds of transistors at most) to get a certain level

of accuracy. Moreover, the verification of CPSs requires accurate fault and system models, as well as the use of comprehensive methods of analyzing the impact of faults in each of the components. This is only possible with an analysis method that works at several levels of abstraction. However, it is unlikely that the existing single verification techniques will suffice at every level. A collection of techniques with suitable integration of the results is required.

There has been much interest in developing formal verification frameworks to verify the correctness of the implementation of CPSs, at different abstraction levels. These formal verification techniques are very efficient in providing guarantees about the model correctness, as well as locating corner-cases and hard-to-find bugs. For example, different formal verification frameworks were developed to verify the implementation of pacemaker’s systems. In the work proposed by [66, 75, 76, 131], a model-based framework for the automatic verification of the functionality of cardiac pacemakers was developed. The authors developed a detailed model of a basic dual chamber pacemaker. This model is constructed based on the timed automata (TA) of each of the pacemaker sub-components. Moreover, in this work, the authors have developed a TA of the heart behavior. The functionality of the pacemaker model has been verified using statistical model checking. However, existing formal-based techniques are designed to detect implementation bugs in the CPS (i.e., identification of functional errors) [121, 122]. In other words, such techniques assume that the CPS always operates in an error free environment. Therefore, with these techniques, it is not possible to detect non-functional faults, such as radiation-induced soft-errors. These errors are often generated when a sensitive area of a CPS is hit by a strong enough flux of external ionizing radiation, such as X-rays and Gama-rays. These external radiation fluxes may change the output of a transistor for a short period of time, which, in turn, may the value stored in a state element. This event is known as a Single Event Upset (SEU).

1.1.1 Radiation Ground Testing

The traditional and most direct approach to evaluate the SEU vulnerability of a system (i.e., an application running in a processor) is through a process called *dynamic radiation ground testing* [26, 133]. This method consists in exposing the target system to a radiation flux and counting the number of errors observed. The outcome is

computed in the form of a parameter known as the *dynamic cross section* (σ), which is defined as the ratio between the number of errors observed at the output of a design configured into the SRAM-based FPGA, divided by the fluence of hitting particles [116]. A problem with that metric is that any change in the application requires a new dynamic test, thus resulting in a very expensive and time-consuming method. Alternative methods for SEU estimation have emerged, with the goal of reducing the time and cost constraints associated with *dynamic radiation ground testing*. In [16, 117, 135], the authors introduce a method of injecting SEUs at random time intervals through emulation, by making use of an interrupt routine to alter values within the processor’s internal registers and memory. Fault injection through emulation is also used in the *direct memory access SEU emulation* method [55], where a dedicated hardware component, controlled externally, selects the time instant and the bit to be altered in the memory. This approach is further explored in [56, 57], where the SEU injection is performed through probabilistic models of the system, with the goal of estimating the system’s time to failure (TTF) and time to recover (TTR). However, this technique still requires emulation in order to obtain certain system rates which the model is built upon (i.e., coverage factor, error factor, and failure factor).

1.1.2 Radiation Testing through Simulation

Another branch of SEU estimation techniques focus on fault injection through simulation, which is usually done by injecting faults at logical or electrical levels [48, 64, 83, 85]. The advantage of these techniques is the high level of control over the fault injection scenarios, since the user has free access to the entirety of the system and the timing of the injections is very accurate. However, emulation and simulation based techniques have severe drawbacks. Disregarding the considerable time required to simulate or to emulate a scenario of thousands of injected faults [17], both approaches are limited in terms of accuracy. This problem arises due to the fact that these techniques are not exhaustive (only consider a small subset of the possible fault injection scenarios) [84].

1.1.3 Radiation Testing through Formal Verification

Recently, the use of *formal based techniques* to analyze soft errors at logical and higher abstraction levels has been proposed, such as the work done in [15, 23, 101]. These techniques provide new insights into the vulnerability of digital designs to SEUs. This is mainly because they are exhaustive and not limited by the number of test vectors as in simulation based techniques. However, at logical abstraction level, these techniques suffer from the *state explosion problem* [65]. Therefore, it is expected for these techniques to be more efficient at higher abstraction levels, such as system-level.

The need to take the complete digital system design into consideration for quantitative safety analysis has led to the widespread acceptance of Fault Trees (FTs). FTs are top-down graphical representations of various combinations of lower level events that may cause the system to reach a top level failure (i.e., system failure) [138]. Fault Tree Analysis (FTA) can provide insightful information to designers regarding the reliability of their systems, such as how is their system most likely to fail and what are the most efficient ways to make it safer. However, the traditional way of conducting FTA is either through paper and pencil proof or through computer simulation techniques, which are inefficient and prone to inaccuracy. The most prominent use of FTA in the literature is by converting the system's FT into a Boolean function and simulating that function with different low level component failure rates [35, 51, 124]. However, these approaches are also costly in time and resources. This is mainly due to the fact that their modeling of FT is limited to the Boolean representation, which may exponentially increase the resource requirements to reach the desired results.

1.2 Thesis Objectives

Since the vast majority of techniques found in the literature rely heavily on low-level testing (i.e., after manufacturing) the main scientific objectives of this thesis are to provide practical frameworks to evaluate and improve CPS reliability at early stages of development and to reduce the complexity of the reliability analysis while improving the accuracy of the results. Due to technological limitations, however, the work presented in this thesis is limited to the analysis of the *main processing units* of CPSs. Namely, this thesis seeks to provide answers for the following questions:

- **Q1:** *How to accurately model a CPS processing unit, at high-level of abstraction?*
- **Q2:** *How to abstract the SEU propagation behavior observed at transistor level at high-levels of abstraction?*
- **Q3:** *How to efficiently utilize formal verification methods to model and analyze SEU vulnerability at high-levels of abstraction?*
- **Q4:** *How to utilize formal verification methods to evaluate and to propose improvements to the system design?*
- **Q5:** *How to measure the vulnerability of complex designs, at high-level, without losing the accuracy provided from the low-level analyses?*
- **Q6:** *Is it possible to improve scalability while preserving accuracy ?*

1.3 Thesis Contributions

In this work, I propose and develop a solution to the aforementioned issues found in the literature. A multi-phased cross-layer methodology is proposed to compute an accurate early estimation of a design’s vulnerability to errors. This methodology is also able to identify and isolate the most critical components of the design, and to pinpoint the most efficient ways to mitigate the design’s weaknesses through the implementation of architectural mitigation, such as TMR. The methodology is also able to investigate the source of a component’s vulnerability and to propose the implementation of redundancy inside of the critical component, if that option is deemed to be the most beneficial. This is achieved by investigating the dependability results obtained from the Fault Tree Analysis (FTA) and the component-level analysis of a design. This work includes the construction of a library of models of FT gates and relationships, as well as a library of models of component-level elements (i.e., registers, ALU, logic gates, etc.). Much like classes in an object-oriented language, these libraries can be used to generate any FT diagram and any component-level description needed. The analysis of these elements is performed through Probabilistic Model Checking (PMC) and Stochastic Model Checking (SMC), both of which provide automatic investigation of the design at multiple levels of detail. To the best of our

knowledge, the above capabilities are not shared by any other existing vulnerability estimation technique. Furthermore, those capabilities will allow us to provide a more comprehensive study of the system and its vulnerability to the designers, before the system is manufactured.

The analysis methodology proposed in this thesis follows cross-layer approach, with the purpose of improving the scalability and efficiency of the modeling as well as of the analysis of the reliability of CPSs. The methodology is composed by two core phases: the fault-tree analysis phase and the component-level analysis phase, as depicted in Figure 1.

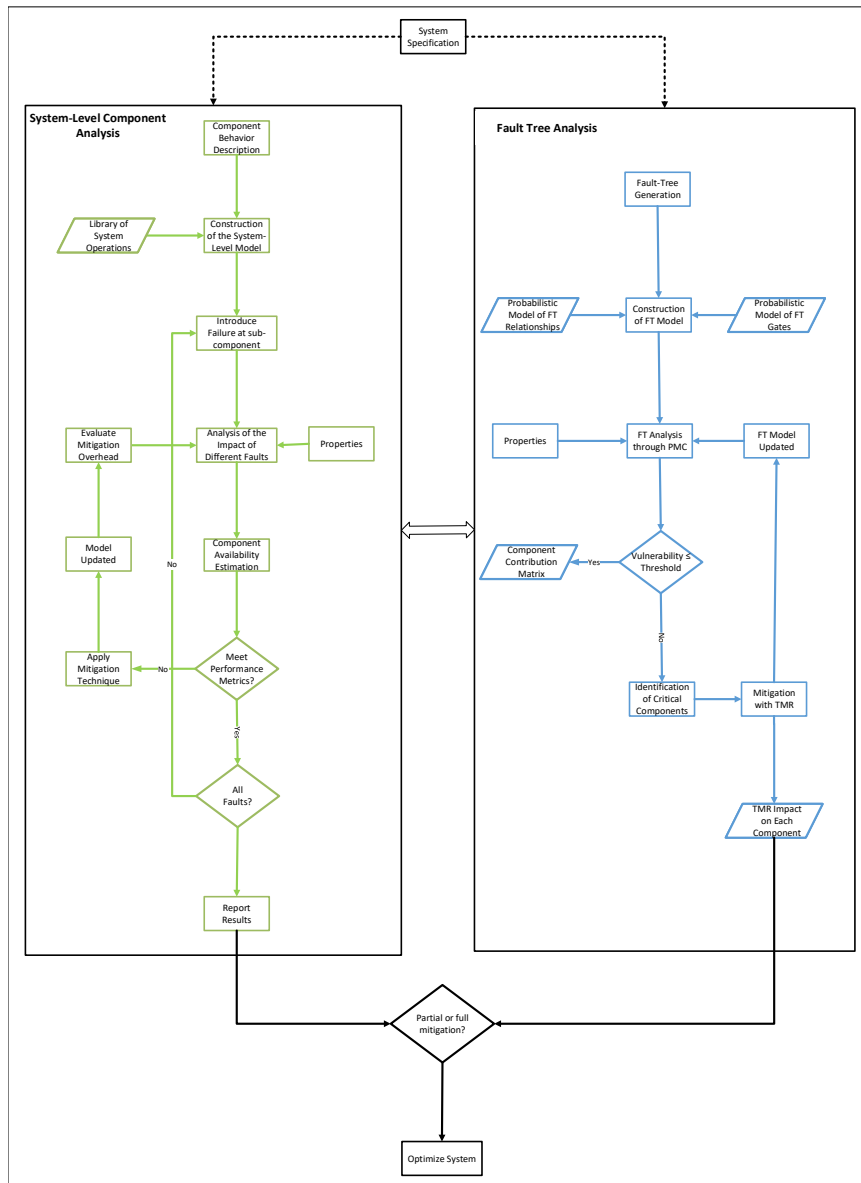


Figure 1: Overview of the Proposed Framework

1.3.1 Fault-Tree Analysis Phase

At the highest level of abstraction, the design's vulnerability is evaluated through FTA. The objective of this phase is to conduct an analysis on the complete system design. This provides an estimation of the probabilistic failure rates of the system, the probabilistic failure rates of each individual component, the fault propagation rates between different components (i.e., how different components impact each other as well as how they impact the overall system vulnerability), and the most relevant

causes of failure to be considered. This phase include the following steps:

- First, a fault tree of the design is generated from the high-level system specifications. This can be done either manually or through special tools, such as the tool introduced in [99], which is able to synthesize the fault tree of a design from its SysML model representation [61]. At this stage, each component of the system is treated as a black box.
- Next, the probabilistic model of the fault tree is constructed from the fault tree diagram representation. This fault tree model is constructed in the language of the probabilistic model checker to be used (in this case, PRISM). The fault tree model is build from our existing libraries of FT gates and relationships.
- A set of properties is derived from the system specifications, denoting how the system is expected to perform.
- With the fault tree model and the properties ready, probabilistic model checking is performed in order to evaluate the design’s vulnerability to faults, as well as the impact of each component to the system vulnerability.
- The impact of each component to the vulnerability of the system is recorded in a library called *Component Contribution Matrix (CCM)*.
- If the system’s estimated vulnerability is below the accepted threshold, the analysis stops. However, if the system’s vulnerability is higher than the threshold, a new step is performed, in which the most critical components of the system are identified within the CCM.
- Next, the components classified as the most critical are isolated and a study is performed on them with the goal of identifying the best TMR configuration to help mitigating the component’s vulnerability.
- The final step in this phase is to record the impact of the different TMR configurations on each critical component, and to update the fault tree model with the most efficient TMR configuration, followed by a new analysis with probabilistic model checking.

Many different modeling and analysis advancements have been incorporated in the fault-tree analysis phase of the proposed methodology. The problem tackled in this phase encapsulates questions *Q1*, *Q2*, *Q4* and *Q5*, presented in Section 1.2. The contributions of this work to the state-of-the-art are the following:

1. **Investigate the impact of different stochastic models on the analysis of a system:** The modeling methodology is applied to a case study of a solar array mechanical system. Next, multiple experiments are conducted on the same system, first by modeling it using Discrete-Time Markov Chain (DTMC) to model known environment scenarios where the probabilistic distribution of the system's behavior is known, and then using Markov Decision Process (MDP) to model the non-deterministic behavior of the system when subjected to unknown environments. These analyses have led to the following publication:

C1: Ammar, M., Hoque, K.A., Ait Mohamed, O. Formal analysis of fault tree using probabilistic model checking: A solar array case study. In Annual IEEE Systems Conference (SysCon 2016).

2. **Investigate system vulnerability to soft-faults and how to efficiently mitigate it:** Redundant architectures, such as Triple Modular Redundancy (TMR), are broadly used as alternatives for fault tolerance, in order to improve the reliability of safety-critical systems. However, the type and placement of the redundant architecture may have a significant impact on the outcome. The experiments that I have conducted demonstrate that redundancy may have negative impacts on the system, in some cases. To avoid issues, an early-analysis is required to determine the type and location of the fault mitigation option. This analysis resulted in the following publication:

C2: Ammar, M., Bany Hamad, G., Ait Mohamed, O., Savaria, Y. Efficient probabilistic fault tree analysis of safety critical systems via probabilistic model checking. In Forum on Specification and Design Languages (FDL 2016).

3. High-level modeling and vulnerability analysis of complex real systems over time: Fault-trees are an excellent tool to investigate fault propagation in the system, identifying the ways in which the system is most likely to fail. However, most FTA approaches are not suitable for safety-critical analysis, since current FT modeling techniques cannot capture sequences of actions. Moreover, the binary representation of FTs is not adequate for systems with complex state-spaces. Finally, FTA is unable to predict the state of the system over a period of time. To solve these shortcomings, I have proposed a new fault-tree paradigm entitled Temporal-Dynamic Fault Trees (TDFTs). I have proposed a new analysis method, based on stochastic model checking, and new FT gate models that evolve over time and that are sensitive to temporal events, such as SEUs. The proposed modeling and analysis advance the state-of-the-art, allowing the power and efficiency of FTs to be used in predictive system evaluation, as well as in significantly more accurate vulnerability assessment to soft-errors. These advances have resulted in the following publications:

C3: *Ammar, M., Bany Hamad, G., Ait Mohamed, O. and Savaria, Y., 2018, December. Reliability Analysis of the SPARC V8 Architecture via Fault Trees and UPPAL-SMC. In 2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS) (pp. 437-440).* **J1:** *Ammar, M., Bany Hamad, G., Ait Mohamed, O., and Savaria, Y. Towards an Accurate Probabilistic Modeling and Statistical Analysis of Temporal Faults via Temporal Dynamic Fault-Trees (TDFTs). IEEE Access Volume 7, page(s): 29264 – 29276 DOI:10.1109/ACCESS.2019.2902796, 2019.*

1.3.2 Component-Level Analysis Phase

In this phase, the analysis of vulnerability is focused on each individual component that forms the system design. Unlike the fault tree analysis phase, where the components were treated as black boxes, in this phase each component is further elaborated and analyzed based on its internal architecture. The goal of this phase is to identify the sources of the component's vulnerability and to study the most efficient ways to internally mitigate that vulnerability. This phase include the following steps:

- Obtention of the component's behavior description. For example, if the component to be analyzed is an Arithmetic Logic Unit (ALU), we must obtain information about its architecture, what operations it performs, what is it connected to, etc.
- Next, a library of system operations is built. Much like classes in an object oriented language, this library contains models of the operations (subcomponents) that serve as the building blocks for the component to be analyzed, such as logic gates, adders, multipliers, etc.
- Based on the component's behavior description, operations from the library of system operations are instantiated in order to construct the model of the component at system-level.
- Once the model is ready, an exhaustive fault injection analysis is conducted in order to identify any weaknesses in the component.
- The component, subjected to fault injections, is verified against a set of properties derived from the system specifications. This verification process is done through probabilistic model checking with PRISM. With this analysis, the impact of each fault injection scenario is obtained.
- Next, with the results obtained through the probabilistic analysis, the limit of the component's availability is estimated.
- If the availability of the component does not meet the preestablished performance metrics, TMR mitigation is applied to the most critical subcomponents. In this case, the model is updated to reflect the changes, and an evaluation of the mitigation overhead is conducted. Then, the probabilistic analysis process is repeated.
- If the availability of the component meets the preestablished performance metrics partially (i.e., the component passes the test but some of its subcomponents do not), fault mitigation must be applied to the critical components and the model must be reevaluated, restarting the fault injection process.
- If all the availability metrics are met (component overall, and all the subcomponents), the phase ends and the results are reported.

- Finally, an additional step takes place, in which the mitigation proposed by each of the phases is compared and the most efficient solution is chosen.

The problems addressed by this phase of the methodology are encapsulated in questions *Q1*, *Q2*, *Q3*, *Q5* and *Q6*, presented in Section 1.2. The advancements achieved by the proposed techniques are the following:

1. **Investigate the impact of soft-errors at system-level using PMC:** The proposed system-level approach consists in modeling the system to be tested with all its components and their expected logical behaviors. Then, the fault-injection points are identified and the fault propagation paths are obtained by counter-example generation with PMC. Subsequently, the analysis performed consists in the probabilistic evaluation of several vulnerability metrics, such as Mean Time to Failure and Mean Time To Recover. These metrics are evaluated for each individual type of fault in the system. Furthermore, the analysis computes the contribution of each component of the system to a failure. This idea generated the following publications:

C4: *Ammar, M., Bany Hamad, G., Ait Mohamed, O., Savaria, Y., Velazco, R. Comprehensive vulnerability analysis of systems exposed to SEUs via probabilistic model checking. In IEEE European Conference on Radiation and Its Effects on Components and Systems (RADECS 2016).*

J2: *Ammar, M., Bany Hamad, G., Ait Mohamed, O., Savaria, Y. System-Level Analysis of the Vulnerability of Processors Exposed to Single-Event Upsets via Probabilistic Model Checking. IEEE Transactions on Nuclear Science. 2017 Sep;64(9):2523-30.*

B1: *Ammar, M., Bany Hamad, G., Ait Mohamed, O., Savaria, Y. (2018). System-Level Modeling and Analysis of the Vulnerability of a Processor to Single Event Upsets (SEUs). Velazco R, McMorro D, Estela J. Radiation Effects on Integrated Circuits and Systems for Space Applications.: (pp. 13-38), Springer, 2019. DOI: 978-3-030-04660-6-2.*

2. **Application-based analysis of the impact of soft-errors on a CPS using PMC:** To provide a better estimation of CPS vulnerability, not only the

hardware but also the software (application) must be considered in the analysis. Based on my component-level modeling approach for fault injection and generation of fault propagation paths, I have proposed a new analysis technique to perform PMC on an application execution trace. For each instruction, the propagation of SEUs is modeled as a Continuous-Time Markov Chain (CTMC), based on the hardware's microarchitecture. From these models, a full estimation of the fault propagation probabilities and latency through each instruction is computed. Furthermore, this model allows the analysis of fault propagation probabilities through the entire program execution. This analysis resulted in the following publications:

C5: *Bany Hamad, G., Ammar, M., Ait Mohamed, O., and Savaria, Y. System-Level Characterization, Modeling, and Probabilistic Formal Analysis of LEON3 Vulnerability to Transient Faults. In IEEE European Conference on Radiation and Its Effects on Components and Systems (RADECS 2018).*

J3: *Bany Hamad, G., Ammar, M., Ait Mohamed, O., and Savaria, Y. New Insights Into Soft-Faults Induced Cardiac Pacemakers Malfunctions Analyzed at System-Level Via Model Checking. IEEE Access. PP. 1-1. 10.1109/ACCESS.2018.2876318, 2018.*

1.4 Thesis Organization

The remainder of this thesis is organized as follows:

Chapter 2 briefly discusses the most relevant SEU vulnerability analysis techniques in the literature.

In Chapter 3, the formal verification methods and tools utilized in this thesis to model and analyze the propagation of SEU at high-level abstraction are introduced.

Chapter 4 explains the basics of FT modeling and analysis with PMC. The introduced PMC-based methodology for FTA modeling is easily expandable with state-efficient models that are modular and constructed by parallel composition. The modeling methodology is applied to a case study of a solar array mechanical system, first using

Discrete-Time Markov Chain (DTMC) to model known environment scenarios where the probabilistic distribution of the system’s behavior is known, then using Markov Decision Process (MDP) to model the non-deterministic behavior of the system when subjected to unknown environments. This work focuses on evaluating and characterizing fault propagation in FTs.

Chapter 5 first introduces a technique, based on FTA and PMC, to evaluate fault mitigation through triple modular redundancy. This approach consists of modeling the behavior of each FT gate as a probabilistic automaton (PA). Thereafter, a Markov Decision Process (MDP) model of the system’s FT is obtained by the parallel composition of all the PAs that compose the FT. In the analysis step, a Component Contribution Investigation (CCI) is performed. The CCI consists in the evaluation of the contribution of the failure of each subcomponent to the system’s failure. Based on the obtained data, the impact of different types of TMR is evaluated, determining the best location and TMR configuration for the system under analysis.

Chapter 6 presents a new system-level approach to compute an accurate estimate of a processor’s vulnerability to SEU propagation. The propagation of SEUs is modeled as a Continuous-Time Markov Chain (CTMC). Furthermore, probabilistic model checking is utilized to exhaustively estimate the impact of SEUs on the system’s behavior. The proposed CTMC model is analyzed for different SEU injection scenarios and different bit-flip rates. Such analysis is capable of producing an accurate estimation of different reliability metrics, such as Mean Time to Failure (MTTF), Mean Time to Recover (MTTR), and the probability of failure for each SEU injection scenario in the system’s subcomponents. Finally, the chapter shows how the proposed probabilistic system-level analysis can also investigate the optimal self-repair rate required in the system to obtain the desired level of availability.

Chapter 7 introduces a new DFT approach to compute an accurate estimation of a system’s vulnerability to soft-faults in complex real-world systems, using a new modeling of FT gates that is based on the Priced-Timed Automata theory. The chapter discusses the modeling of FTs of complex systems and it is centered around a case-study of the 32-bit SPARC V8 integer pipeline. The analysis is fully automatic,

conducted through stochastic model checking, with UPPAAL-SMC. The analysis presented consists in the estimation of the probability of each type of Trap Exception (TE) to occur in the targeted architecture, as well as the impact of individual registers to the overall reliability of the processor, and the probability of failures over time. Chapter 8 proposes a new fault tree modeling paradigm, to capture the impact of temporal events in systems, called *Temporal Dynamic Fault Trees (TDFTs)*. The proposed TDFTs are utilized to model fault propagation in complex systems while conserving the characteristics and dependencies between different temporal events, soft-faults, and permanent faults. This chapter also introduces a new analysis approach utilizing stochastic model checking with UPPAAL-SMC, which, combined with efficient modeling, is able to circumvent the state-explosion problem that is inherent to other model-checking approaches. The analysis proposed in this chapter is able to evaluate the impact of temporal faults in systems, as well as to estimate the reliability and availability of the system over extended periods of time.

Chapter 9 provides a general discussion about the present work, which has been detailed in Chapters 4 through 8, and finally Chapter 10 summarizes this thesis and proposes some directions for future work.

Chapter 2

Literature Review

The modeling and analysis of SEU relevant faults and their mitigation for dependability analysis is an active research area. The most common approach to evaluate the SEU vulnerability of a system is through a process called *dynamic radiation ground testing* [26, 134]. This method consists in exposing the target system to a radiation flux and counting the number of errors observed. The outcome is computed in the form of a parameter known as the *dynamic cross-section* (σ), which is defined as the ratio between the number of errors observed at the output of a Design Under Test (DUT), divided by the fluence of hitting particles [116]. A problem with that metric is that any change in the application requires a new dynamic test, thus resulting in an expensive and time-consuming method.

Alternative methods for SEU estimation have emerged, with the goal of reducing the time and cost constraints associated with *dynamic radiation ground testing*. In [117, 135], the authors introduce a method of injecting SEUs at random time intervals through emulation, by making use of an interrupt routine to alter values within the processor's internal registers and memory. Fault injection through emulation is also used in the *direct memory access SEU emulation* method [55], where a dedicated hardware component, controlled externally, selects the time instant and the bit to be altered in the memory. This approach is further explored in [56, 57], where the SEU injection is performed through probabilistic models of the system, with the goal of estimating the system's time to failure (TTF) and time to recover (TTR). However, this technique still requires emulation in order to obtain certain system rates which the model is built upon (i.e., coverage factor, error factor, and failure factor). Another

branch of SEU estimation techniques focus on fault injection through simulation, which is usually done by injecting faults at logical or electrical levels [64, 83, 85]. The advantage of these techniques is the high level of control over the fault injection scenarios, since the user has free access to the entirety of the system and the timing of the injections is very accurate. However, emulation and simulation based techniques have severe drawbacks. Disregarding the considerable time required to simulate or to emulate a scenario of thousands of injected faults [17], both approaches are limited in terms of accuracy. This problem arises due to the fact that these techniques are not exhaustive, but rather reliant on input vectors [84]. These approaches are also reliant on detailed models of the DUT, which are not always available.

Recently, the use of *formal based techniques* to analyze soft errors at logical and higher abstraction levels has been proposed, such as the work done in [23]. These techniques provide new insights into the vulnerability of digital designs to SEUs. This is mainly because they are exhaustive and not limited by the number of test vectors as in simulation based techniques. In formal techniques, the user starts out by stating what output behavior is desirable and then lets the formal checker prove or disprove it. In other words, given a property, formal verification exhaustively searches all possible input and state conditions for failures. However, at logical abstraction level, these techniques suffer from *state explosion* (i.e., exponential growth in the number of states of the model) [65]. Therefore, it is expected for these techniques to be more efficient at higher abstraction levels, such as system-level.

In recent years, a number of works related to fault tree analysis of SEUs has emerged. In [14] a tool for fault tree analysis is presented, called DFTCalc, which is capable of modeling fault trees via compact representations. That work uses the stochastic technique to perform the analysis of dependability properties. The advantage of DFTCalc is a more expressive syntax for FTD, but it cannot check the correctness and completeness of fault trees. Moreover, an important difference is that their work does not include the possibility of fault masking, whereas in our work this important phenomenon is taken into account. The work presented in [6] gravitates towards theorem proving, using HOL4 and higher-order logics to analyse safety-critical systems through FTA by formalizing the gates of a fault tree and conducting FTA-based failure analysis. However, a model checking approach has several advantages over theorem proving, such as being systematically exhaustive, fully automated, and

more time efficient. However, FTA analysis of SEUs is very limited in the literature, mostly limited to post-error analyses.

Chapter 3









Background Information

3.1 Fault Tree Analysis

The fault tree analysis method was developed in 1962 by Bell Telephone Laboratories and it is a widely used method for risk assessment, mainly in the area of avionics, nuclear and chemical industries [138]. FTA follows a deductive approach, which means that it starts from an undesirable general event in order to find the origins of said event. In the context of FTA, the general event is known as the *top event*, from which the fault tree branches out vertically. The top event is defined as the failing point of a system in operating conditions, whether those conditions are considered normal or abnormal. A single fault tree can be used to analyse one single top event, which can then be fed into another fault tree as a bottom event. Bottom events are the ones at the very bottom of the fault tree, independent from any other events, and, assuming the FTA is performed following a quantitative evaluation, these events receive fault probabilities that will dynamically spread through the rest of the tree during the analysis. The elements of a Fault Tree and their graphical representations are summarized in Table 2.

Other elements compose the fault tree, such as conditioning events, which are specific restrictions applied to a logic gate within the fault tree, like a mode of operation or a sequence of other events that serve as a prerequisite to its activation. These events can prevent or enforce the activation of a node or a gate, according to the circumstances. An external event is an event expected to occur and it is not considered as a fault; as such it may or may not affect the FTA. Lastly, an undeveloped event is an event

Table 1: TMR configurations and Respective FTs

Figure	Event	Type	Description
	Component	na	System component
	Bottom Event	na	Basic event that may lead to a failure
	AND Gate	Static	The output is true if all inputs are true
	OR Gate	Static	The output is true if at least one input is true
	Combination Gate	Dynamic	The output is true if n inputs are true
	Priority AND Gate	Dynamic	The output is true if all the inputs become true in a specific sequence
	Inhibit Gate	Dynamic	The output is true if the single input becomes true in the presence of an enabling condition
	Spare Gate	Dynamic	The output is true if the component and all its spares fail

about which insufficient information is available or which is of no consequence to the system. Normally these events are overlooked and have no real impact over the FTA. In recent years, a number of works related to fault tree analysis has emerged. In [14] a tool for fault tree analysis is presented, called DFTCalc, which is capable of modeling fault trees via compact representations. That work uses the stochastic technique to perform the analysis of dependability properties. The advantage of DFTCalc is a more expressive syntax for FTD, but it cannot check the correctness and completeness of fault trees. Moreover, an important difference is that their work does not include the possibility of fault masking, whereas in our work this important phenomenon is taken into account. The work presented in [6] gravitates towards theorem proving, using HOL4 and higher-order logics to analyse safety-critical systems through FTA by formalizing the gates of a fault tree and conducting FTA-based failure analysis. However, a model checking approach has several advantages over theorem proving, such as being systematically exhaustive, fully automated, and more time efficient. In [124], authors propose the use of Binary Decision Diagrams (BDD) to perform quantitative analysis in fault trees. Their method improves the accuracy of the calculated failure rates of bottom events over simulation techniques. It consists of converting the FTD into a format compatible with Shannon’s decomposition, allowing the failure rates to be accurately calculated. The binary decision diagram approach is extended to qualitative FTA analysis, in [125], where BDDs are employed to evaluate minimal cutsets of a fault tree without creating probabilistic inaccuracies like in the conventional qualitative analysis techniques.

3.2 Triple Modular Redundancy

One of the most important architectural patterns used in safety engineering is the Triple Modular Redundancy (TMR) [4, 70, 79, 93]. The idea of TMR consists in triplicating a module that is considered critical in order to guarantee a correct behavior of the system. As shown in Figure 2 [33], the input is replicated to each copy of the module M , and the output is provided to a voter V whose role is to propagate the value that is in accordance with the majority of M outputs. The impact of a Triple Modular Redundancy approach is to increase the reliability when compared with a single module. In other words, the main goal is to decrease as much as possible the gap with respect to a perfect (faultless) component. This concept drives the evaluation of redundant architectures.

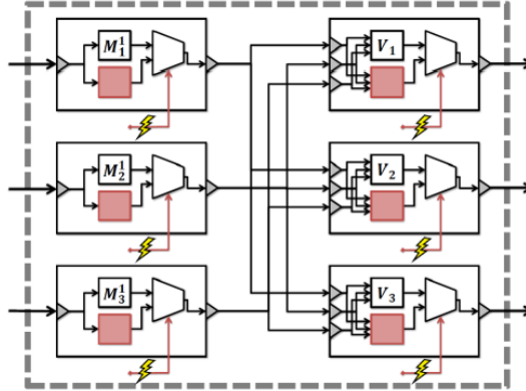


Figure 2: Example of TMR Applied to a Generic Component

Redundant architectures exploiting Triple Modular Redundancy (TMR) are broadly used to obtain fault tolerant safety-critical systems. In the literature, the utilization of formal methods to analyze these architectures is rather limited. In [?] the behavior of a single TMR is formalized through Communicating Sequential Processes (CSPs). The work in [152] uses Uppaal model checker to analyze a timed automata model of a TMR system design. Both techniques are limited to a single system and do not consider multi-staged TMRs. In [33, 34], the fault tree of a redundant system is modeled and analyzed through Satisfiability Modulo Theories (SMTs), giving an estimation of the reliability gain across different TMR configurations. The main problem of these approaches is the amount of redundancy required to perform the analysis. Furthermore, their proposed modeling has the same limitation as previous approaches because they also rely on boolean representation. Due to the increased

cost and physical size constraints, the main challenge of implementing fault tolerance through TMRs is knowing where in the system to apply the redundancy and knowing which TMR configuration is the most beneficial to the overall reliability.

3.3 Probabilistic Model Checking

Probabilistic model checking is a formal verification technique, derived from regular model checking, applied on systems that present a random or probabilistic behavior, like real life applications, where the resulting models usually contain a very large number of states. It would be very impractical for the user to explicitly model every state and transition of these applications. The power of model checkers comes from the exhaustive nature of the analysis they perform. The model checker is capable of traversing all reachable states in the model in order to generate a solution. One of the most notable features of this technique is not only being able to receive probabilities as an input but also to return probabilities as output. Given that a perfect, bug-free system is something near impossible to achieve, probabilistic model checking allows the user to work with tolerance percentages rather than absolute values.

Examples of the use of probabilistic model checkers are widely found in the literature. In [145] the authors assess the feasibility of using model checking for verification of Unmanned Aircraft Systems (UAS) in civil airspace. The authors begin by modeling simple UAS systems into the SPIN tool and then refining the model by incorporating probabilities and using probabilistic model checking with PRISM. Lastly, they model the UAS using the autonomous agent language Gwendolen and compare and contrast the various approaches. The work in [72] uses PRISM tool and language to perform the formal modeling and verification of RAM related properties on satellite systems, using Erlang distribution to improve discrete time delays in CTMC by approximating nonexponential holding times with intermediate states based on a phase type distribution.

3.3.1 PRISM Model Checker

PRISM [91] is a free, open source probabilistic symbolic model checker developed at the University of Birmingham. It works with its own high-level modeling language, based on the Reactive Modules formalism [7], which is written in form of state-based

modules, each composed by a set of guarded commands. PRISM uses Binary Decision Diagrams (BDD) (Binary Decision Diagrams) and Multi-Terminal Binary Decision Diagrams (MTBDD) [88] to construct and compute the reachable states of even very large probabilistic models.

PRISM is a very flexible tool to work with probabilistic real-life models as it allows for the specification of probabilities inside the model and in the properties. Additionally the software will inform what's the probability of given property failing after the verification. PRISM allows for step-by-step simulation where the user may chose which variables on the system he wants to manipulate as well as their initial values. The simulation may be guided, where the user manually selects the next step to be taken, or random, where the user selects the number of random steps the program should simulate. The tool supports a wide range of model analysis methods and it features a very efficient implementation, making use of a symmetry reduction technique [89] to help mitigate state-space explosion. PRISM works by creating a probabilistic model of the system and computing its reachable states. The model checking is done by dynamically creating graph-based computations [97] in order to reach a numerical solution (based on linear equation systems and optimization problems).

3.3.2 UPPAAL-SMC

UPPAAL is a toolbox for verification of real-time systems, represented by a network of timed automata, extended with integer variables, structured data types, and channel synchronization. For the efficient analysis of probabilistic performance properties, UPPAAL-SMC proposes to work with *Statistical Model Checking* (SMC). SMC works by monitoring some simulations of the system, and then use statistical results (including *sequential hypothesis testing* or *Monte Carlo simulations*) to decide whether the system satisfies some property with a sufficient degree of confidence. The modeling formalism of UPPAAL-SMC is based on a stochastic interpretation and an extension of the *Timed Automata* (TA) formalism used in the classical model checking version of UPPAAL. For individual TA components, the stochastic interpretation replaces the non-deterministic choices between multiple transitions enabled by probabilistic choices (that may or may not be user-defined). Similarly, the nondeterministic

choices of time delays are refined by probability distributions, which at the component level are given either uniform distributions in cases with time-bounded delays or exponential distributions in cases of unbounded delays [46].

An illustrative example of the UPPAAL formalism is given by the PTA in Fig. 34. In this example and through the rest of this paper, the weight annotations on locations and edges are ignored and defaulted to “1”. For Fig. 34, the delay distribution determined by the upper and lower paths to the END state is given by sums of uniform distributions, where $X \geq 2$ (green label) is the guard of the transition (i.e., minimum time), and $X \leq 4$ (purple label) is the invariant distribution (i.e., maximum time delay) of the transition. The stochastic choice that determines which path will be taken is represented by a forked transition, where each path is weighted accordingly. In the example, the weights of each path are either $\frac{1}{6}$ or $\frac{5}{6}$. Finally, an update may be performed during each transition (blue labels). Therefore, the END location in the example is reachable within the interval $X = [4, 12]$.

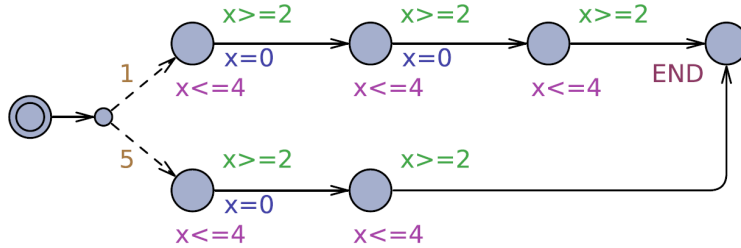


Figure 3: Illustrative Example of the UPPAAL Formalism. Reproduced from [46].

Chapter 4

Article I: Formal Analysis of Fault Tree using Probabilistic Model Checking: A Solar Array Case Study

Authors: Marwan Ammar, Khaza Anuarul Hoque, Otmane Ait Mohamed

Abstract: Fault Tree Analysis (FTA) is a widespread technique used to assess the reliability of safety-critical systems. The traditional way of conducting FTA is either through paper and pencil proof or through computer simulation techniques, which are inefficient and prone to inaccuracy. In this paper, we propose the use of probabilistic model checking to automatically analyze fault trees of safety-critical systems. Our methodology consists in the probabilistic formalization of the gates used in a fault tree to a Discrete-Time Markov Chain (DTMC) and a Markov Decision Process (MDP), and the subsequent probabilistic verification using PRISM tool to quantitatively analyze the system. To illustrate the proposed approach we perform the fault tree analysis of a solar array system, used as power source for the DFH-3 satellite. The results show that harsh thermal environment is the main cause of system failures.

4.1 Introduction

Fault Tree Diagram (FTD) is a top-down graphical model of a system, which represents all paths and events that may lead to failure within that system [138]. Events in FTDs are nodes, connected through logic gates in such a way that an error in one of the bottom nodes can propagate to the higher level nodes and reach the top-level event, compromising the functionality of the entire system. Fault Tree Analysis (FTA) is the study of such diagrams in order to discover and assess the effect of undesirable events or faults [138]. FTA allows safety and reliability engineers to better understand how the system can fail, identifying the best possible ways to make it safer, as well as the system's event rates. FTA is commonly used in the aerospace industry for both hardware and software [138] as means for investigating a system's modes, potential faults occurrences with their causes, and to quantify their contribution to system unreliability in the course of product design. Traditionally, FTA is based on simulation techniques [96, 114, 138], with the main techniques being: Monte-Carlo simulation, Quasi-Monte-Carlo method, time-sequential simulation, and discrete event simulation [53]. Assessing the causes and the probability of a punctual failure occurrence in the system using simulation-based techniques is very costly, since each failure condition must be evaluated separately, one at a time, creating a very large state-space and requiring tremendous effort to analyze the whole scope of the system.

An alternative to avoid the aforementioned problem is the use of Probabilistic Model Checking (PMC). PMC is a formal verification method that designates a collection of techniques for the automatic analysis of reactive, finite state concurrent systems. This technique has several advantages over simulation. Notably, probabilistic model checking is an exhaustive, accurate, efficient and completely automated verification technique [20], providing a comprehensive and reliable solution for fault tree analysis. In this work, we propose a PMC-based methodology for FTA modeling, using PRISM language [1]. Added to the inherent advantages of PMC listed above, PRISM's modularity allows for easily expandable, state-efficient models. The modeling methodology is applied to a case study of a solar array mechanical system [147], first using Discrete-Time Markov Chain (DTMC) [106] to model known environment scenarios where the probabilistic distribution of the system's behavior is known, then using Markov Decision Process (MDP) to model the non-deterministic behavior of the system when subjected to unknown environments.

Our goal is to provide a way for developers to evaluate the weaknesses of their systems. Through PMC properties, our approach can be used to ascertain not only correctness, but also quantitative measures such as performance and reliability, without the time and intensive processing required by simulation techniques. This work focuses on evaluating the dynamics of fault propagation in FTDs, and we observe that the lowest levels in the tree are the most positively affected by fault masking, events connected to multiple gates are especially unreliable in non-deterministic environments (MDP), and that events connected to *AND* gates are more affected by non-determinism than events connected to *OR* gates.

The following section presents some important background information about PMC, the PRISM tool and FTA. Section 4.3 considers related works. In Section 5.3.1 we describe our modeling approach. In Section 5.4 we demonstrate the application of our proposed methodology, performing the analysis on a solar array FTD and Section 5.5 concludes the paper with some future research directions.

4.2 Preliminaries

4.2.1 Probabilistic Model Checking with PRISM

Probabilistic model checking is a formal verification technique derived from regular model checking and applied on systems that present a random or probabilistic behavior, like real life applications, where the resulting models usually contain a very large number of states. This technique can deal with a wide range of quantitative measures; the results show an exact figure of the property being verified, usually in parts-per-hundred; it can be fully automated, provides an exhaustive analysis of the model, and it is very efficient. PMC works with several model types and temporal logic specification languages. In this paper we use DTMC and MDP as the model types and Probabilistic Computation-Tree Logic (PCTL) as the property specification language. In [60] DTMC is defined as a tuple $D=(S, \bar{s}, P, L)$ where S is a countable set of states, $\bar{s} \in S$ is an initial state, $P : S \times S \rightarrow [0, 1]$ is a transition probability matrix such that $\sum_{s' \in S} P(s, s') = 1$ for all $s \in S$, and $L : S \rightarrow 2^{AP}$ is a labeling function mapping each state to a set of atomic propositions taken from a set AP. An MDP is defined in [60] as a tuple $M = (S, \bar{s}, \alpha_M, \delta_M, L)$ where S is a finite set of states, $\bar{s} \in S$ is an initial state, α_M is a finite alphabet, $\delta_M : S \times \alpha_M \rightarrow Dist(S)$ is

a (partial) probabilistic transition function and $L : S \rightarrow 2^{AP}$ is a labelling function mapping each state to a set of atomic propositions taken from a set AP. As such, the MDP is a stochastic system where all the decisions are made in a non-deterministic manner.

To complete the model checking process, we specify properties using a probabilistic extension of CTL temporal logic called PCTL. PCTL can be used with both DTMC and MDP models, working at discrete time domain. The main difference is that using PCTL over MDP model requires to extend the $P[]$ (probability query) operator with the *min* and *max* operators. As such, each path formula is evaluated in a best or worse case scenario [12]. Below are two illustrative examples with their natural language translation:

1. DTMC - $P_{>0.8} [\neg a \cup b]$ - “The probability of a being false *until* b is true is bigger than 0.8.
2. MDP - $P_{min} [F (a > 0) \cup \neg b]$ - “What’s the minimum probability that *eventually* a will be bigger than zero *until* b is false.












We perform PMC with PRISM [1], a free, open source probabilistic symbolic model checker. It works with its own high-level modeling language, written as state-based modules. Each module is composed by a set of guarded commands. PRISM supports a wide range of model analysis methods and it features a very efficient implementation, making use of multiple model checking engines (based on BDDs and their extensions). These engines enable PRISM to handle models with up to 10^8 states. The model checking is done by dynamically creating graph-based computations [100] in order to reach a numerical solution based on linear equation systems and optimization problems. PRISM also features advanced algorithms such as symmetry reduction and abstraction refinement. The syntax of the PRISM language as well as some examples will be given in Section 5.3.1.

4.2.2 Fault Tree Analysis

The fault tree analysis method was developed in 1962 by Bell Telephone Laboratories and it is a widely used method for risk assessment, mainly in the area of avionics, nuclear and chemical industries [138]. FTA follows a deductive approach, which means

that it starts from an undesirable general event in order to find what circumstances may lead to that event. In the context of FTA, the general event is known as *top event*, from which the fault tree branches out vertically. The top event is defined as the failing point of a system in operating conditions, whether those conditions are considered normal or abnormal. A single fault tree can be used to analyse one single top event, which can then be fed into another fault tree as a bottom event. Bottom events are the ones at the very bottom of the fault tree, independent from any other events, and, assuming the FTA is performed following a quantitative evaluation, these events receive fault probabilities that will dynamically spread through the rest of the tree during the analysis. The elements of a Fault Tree and their graphical representations are summarized in Table 2.

Table 2: Elements of a Fault Tree

Token	Element	Description
	Top or Intermediary Event	System or component failure
	Bottom Event	A basic initiating fault event
	Conditioning Event	Specific condition or restriction that can apply to any gate
	External Event	Event that is normally expected to occur
	Undeveloped Event	Event that's not further developed due to lack of importance or knowledge
	AND Gate	The output is true if all inputs are true
	OR Gate	The output is true if at least one input is true
	Combination Gate	The output is true if n inputs are true
	Exclusive OR Gate	The output is true if exactly one of the inputs is true
	Priority AND Gate	The output is true if all the inputs become true in a specific sequence
	Inhibit Gate	The output is true if the single input becomes true in the presence of an enabling condition

4.3 Related Works

PRISM model checker has several application domains, specially for safety critical systems. In [145] the authors assess the feasibility of using model checking for verification of Unmanned Aircraft Systems (UAS) in civil airspace. The authors begin by modeling simple UAS systems into the SPIN tool and then refining the model by incorporating probabilities and using probabilistic model checking with PRISM.

Lastly, they model the UAS using the autonomous agent language Gwendolen and compare and contrast the various approaches. The work in [72] uses PRISM tool to perform the formal modeling and verification of RAM related properties on satellite systems, using Erlang distribution to improve discrete time delays in CTMC by approximating nonexponential holding times with intermediate states based on a phase type distribution.

In recent years, a number of works related to fault tree analysis has emerged. In [14], DFTCalc tool for fault tree analysis is presented. It is capable of modeling fault trees via compact representations and the dependability analysis is performed using stochastic techniques. DFTCalc allows modelling of most FTD constructs but it cannot check the correctness and the completeness of fault trees. Moreover, an important difference is that their work does not include the possibility of fault masking, whereas in our work this important phenomenon is taken into account. The work presented in [6] gravitates towards theorem proving, using HOL4 and higher-order logics to analyse safety-critical systems through FTA by formalizing the gates of a fault tree and conducting FTA-based failure analysis. However, a model checking approach has several advantages over theorem proving, such as being systematically exhaustive, fully automated, and more time efficient. In [124], authors propose the use of Binary Decision Diagrams (BDD) to perform quantitative analysis in fault trees. Their method improves the accuracy of the calculated failure rates of bottom events over simulation techniques. It consists of converting the FTD into a format compatible with Shannon's decomposition, allowing the failure rates to be accurately calculated. The binary decision diagram approach is extended to qualitative FTA analysis, in [125], where BDDs are employed to evaluate minimal cutsets of a fault tree without creating probabilistic inaccuracies like in the conventional qualitative analysis techniques.

All these works exemplify the versatility and importance of fault tree diagrams and their relevance for assessment of safety-critical systems related to diverse areas. The research presented in this paper is different than what is found in the related work because on top of formalizing the various gates of a fault tree, allowing for the representation of virtually any system, we introduce a modeling technique that is state-efficient and easily scalable. In addition, our modeling can handle both quantitative and qualitative analysis, with a simple change in the PCTL properties being

verified.

4.4 Modeling

In this section we show how the fault tree diagrams are modeled in PRISM. Our emphasis is on *AND* and *OR* gates, since those are the types of gates present in the case study, presented in Section 5.4. The fault-masking mechanic adds a probability of fault mitigation inside the gates, which are designed to allow easy system composition with a reduced number of state transitions. The modeling of the other FTA gates follow a similar approach but are not included in this paper due to space constraints. Since the DTMC and the MDP modeling approaches are similar in PRISM, we will focus our explanation on the DTMC modeling, with graphical representations and a detailed explanation of each gate’s state transitions in Subsections 4.4.1 and 4.4.2. It is important to note that the main difference between DTMC and MDP is that in DTMC, in each state, the successor state is determined by a discrete probability distribution, whereas in MDP, in each state, the successor state is determined by a nondeterministic choice between several discrete probability distributions.

MDP model, in each state S , the successor state is decided in two steps: the first step is non-deterministic and the second step is random, according to the probability distribution of the transition matrix. The DTMC model only has one step, which is the random probabilistic step.

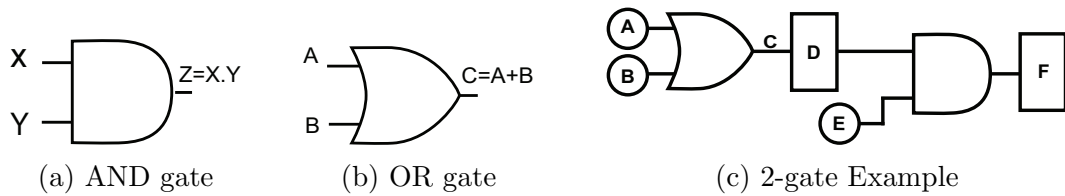


Figure 4: Fault Tree Gates

4.4.1 Modeling of an AND Gate

The *AND* gate is defined as follows:

Definition 1: Given two inputs X and Y and their output Z , connected through an *AND* gate, output Z becomes true if and only if X and Y are true. Figure 14(a)

is a representation of an *AND* gate. The *AND* modeling in PRISM follows these assumptions:

1. All the inputs to the *AND* gate represent events, each of which have a probability of being triggered.
2. Only one input can trigger at a time.
3. If one of the inputs of the *AND* gate is triggered then the other will be given an additional probability of triggering.
4. Before an output is generated, there is a certain probability that the fault will be masked.

The model of the *AND* gate can be defined formally as a finite transition system (S, \bar{s}, P, L) , where S is the set of states $S = (S_0, S_1, S_2, S_3)$, \bar{s} is the initial state $\bar{s} = S_0$, P is a transition probability matrix, P_{ij} , such that $P_{\bar{s}, S_1} = (p_1)$, $P_{\bar{s}, S_2} = (p_2)$, $P_{S_1, S_3} = (p_3)$, $P_{S_2, S_3} = (p_4)$, $P_{S_1, S_4} = (p_5)$, $P_{S_2, S_4} = (p_5)$ and $L : S \rightarrow 2^{AP}$ is a labeling function, mapping states with properties of interest, where $L(S_3) = \text{propagate}$. The DTMC model of the *AND* gate is shown in Fig. 5.

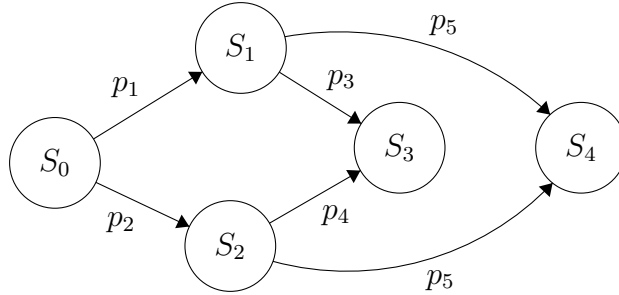


Figure 5: 2-input *AND* gate DTMC

Starting from an initial state $S_0(X=0, Y=0)$ the next state can either be $S_1(X=1, Y=0)$ or $S_2(X=0, Y=1)$, with probabilities $p1$ and $p2$, respectively. At this point, the system can either move to $S_3(X=1, Y=1, Z=1)$, with probability $p3$ or $p4$, signifying fault propagation, or the system can move to $S_4(X=0, Y=0, M=1)$, with probability $p5$, signifying fault masking. The model in Fig. 5 is then encoded into PRISM. A PRISM command is a tuple $cmd = (act, guard, rate, action)$ following the format $[\langle act \rangle] \langle guard \rangle \rightarrow \langle rate \rangle : \langle action \rangle ;$, where act is an action label, $guard$ is a predicate over a variable, $rate$ is a numerical evaluation referent to the probability of

an action and *action* is a set of n variable updates that will translate into transitions in the model. Fig. 6 shows the PRISM representation of the *AND* gate. Please note that variable M stands for *masking* and variable *and* indicates if the module is idle ($\text{and}=0$), waiting for first input ($\text{and}=1$) or waiting for second input ($\text{and}=2$).

```

module and_gate
[] (and=1) & (X=0) & (Y=0) & (M=0) & (Z=0) -> p1 : (X'=1) & (and'=2)
                                     +p2 : (Y'=1) & (and'=2) ;
[] (X=1) & (Y=0) & (M=0) -> p5 : (M'=1) & (X'=0) +p3 : (Y'=1) & (Z'=1) ;
[] (Y=1) & (X=0) & (M=0) -> p5 : (M'=1) & (Y'=0) +p4 : (X'=1) & (Z'=1) ;
endmodule

```

Figure 6: PRISM modeling of an AND gate

4.4.2 Modeling of an OR Gate

The *OR* gate, seen in Figure 14(b), is defined as:

Definition 2: Given two inputs A and B and their output C , output C becomes true if any of the inputs A or B are true.

The modeling process takes into consideration the following assumptions:

1. All the inputs to the gate represent an event, each of which have a probability of being triggered.
2. Only one input can trigger at a time, after which no other input can trigger.
3. Before an output is generated, there is a certain probability that the fault will be masked.

We formally define the *OR* gate as a finite transition system (S, \bar{s}, P, L) , where S is the set of states $S = (S_0, S_1, S_2, S_3)$, \bar{s} is the initial state $\bar{s} = S_0$, P is a transition probability matrix, p_{ij} , such that $P_{\bar{s}, S_1} = (p_6)$, $P_{\bar{s}, S_2} = (p_7)$, $P_{S_1, S_3} = (p_8)$, $P_{S_2, S_3} = (p_9)$, $P_{S_1, S_4} = (p_{10})$, $P_{S_2, S_4} = (p_{10})$ and $L : S \rightarrow 2^{AP}$ is a labeling function, mapping states with properties of interest, where $L(S_3) = \text{propagate}$. The *OR* gate DTMC model is shown in Figure 7.

From an initial state $S_0(A=0, B=0)$ the next state can either be $S_1(A=1, B=0)$ or $S_2(A=0, B=1)$ with probabilities p_6 and p_7 respectively. At this point, the system may either move to $S_3(A=0, B=0 \text{ and } C=1)$, with probabilities p_8 or p_9 , signifying

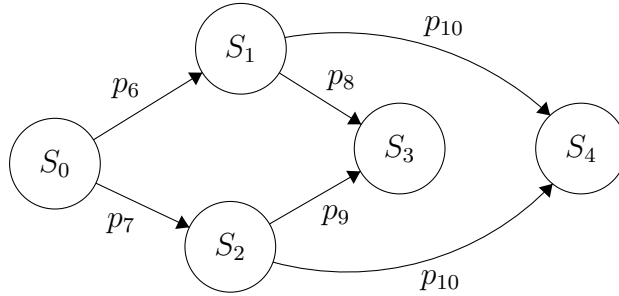


Figure 7: 2 input OR gate DTMC

fault propagation, or to $S_4(A=0, B=0 \text{ and } M=1)$, with probability p_{10} , signifying fault masking. The *OR* gate model is then encoded into PRISM as shown in Fig. 8. The *or* variable is equivalent to the *and* variable, previously explained.

```

module or_gate
[] (or=1) & (A=0) & (B=0) & (M=0) & (C=0) -> p1: (A'=1) & (or'=2)
                                     +p2: (B'=1) & (or'=2);
[] (A=1) & (C=0) & (M=0) -> p5: (M'=1) & (A'=0) +p3: (A'=0) & (C'=1);
[] (B=1) & (C=0) & (M=0) -> p5: (M'=1) & (B'=0) +p4: (B'=0) & (C'=1);
endmodule

```

Figure 8: PRISM modeling of an OR gate

4.4.3 Sample Modeling of a 2-Gate System

To illustrate the modeling process of a fault tree using our modular approach (pre-modeled gates in PRISM), we provide an example using a simple two gates fault tree with three inputs A, B, E , and one output F . A, B and E are bottom events. D is an intermediary event and F is the top event for this example, as shown in Figure 14(c). The assumptions listed in Sections 4.4.1 and 4.4.2 are also applicable for this example, with two additions:

1. There exists one additional module *twogate*, in the PRISM code, that serves as a control module, where the order of the gates in the system can be specified.
2. The system starts at the *OR* gate, where inputs A and B each have a probability of triggering.
3. Output C becomes input D as it enters the *AND* gate.
4. Input E has a chance of being triggered after the output C propagates, following our *AND* gate assumptions.

In this small example, we illustrate how to use the gates, defined in the previous sections, as building blocks to construct more complex fault tree diagrams. Fig. 9 shows the DTMC model of the two-gates example.

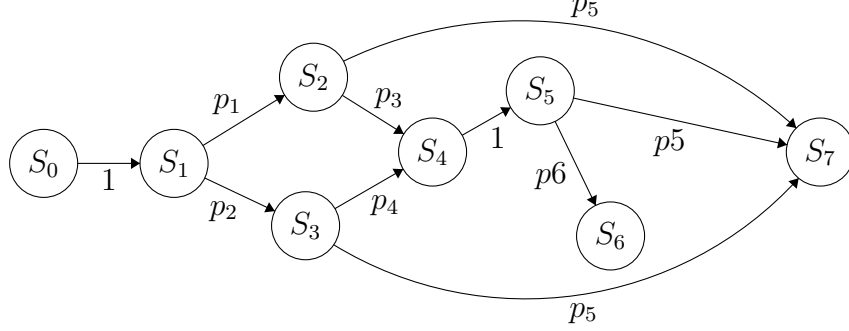


Figure 9: 2-gate DTMC

From state S_0 , the system is initialized and moves to S_1 . In S_1 the system can go to S_2 or S_3 with probabilities $p1$ or $p2$. From S_2 or S_3 , the system can go to S_7 , with probability $p5$, signifying masking, or to S_4 , signifying propagation of the *OR* gate. The output of the *OR* gate serves as one of the inputs of the *AND* gate, thus the system moves to S_5 . In S_5 , if input E is triggered and the output propagates, the system moves to state S_6 , otherwise the system moves to S_7 , signifying masking. The two-gates model is encoded in PRISM as shown in Fig. 10.

```

module twogate
[] or=0 -> (or'=1);
[] c=1 -> (c'=0)&(d'=1);
endmodule
module or_gate
[] (or=1)&(a=0)&(b=0)&(m=0)&(c=0)->p1:(a'=1)&(or'=2)
+p2:(b'=1)&(or'=2);
[] (a=1)&(c=0)&(m=0)->p5:(m'=1)&(a'=0)+p3:(a'=0)&(c'=1);
[] (b=1)&(c=0)&(m=0)->p5:(m'=1)&(b'=0)+p4:(b'=0)&(c'=1);
endmodule
module and_gate
[] (and=1)&(d=0)&(e=0)&(m=0)&(f=0)->p6:(d'=1)&(and'=2)
+p7:(e'=1)&(and'=2);
[] (d=1)&(e=0)&(m=0)->p5:(m'=1)&(d'=0)+p8:(e'=1)&(f'=1);
[] (e=1)&(d=0)&(m=0)->p5:(m'=1)&(e'=0)+p9:(d'=1)&(f'=1);
endmodule

```

Figure 10: PRISM modeling of the 2-gate example

It is important to note that all variables are declared globally, with starting value of zero. The variable declarations are suppressed in Fig. 10 for space constrains. The module *twogate* controls the flow of the system. In the first line of the module, the *or* variable sets the starting point of the system, activating the *OR* gate. The second line of the module takes the output of the *OR* gate, C , and channels it to the input

of the *AND* gate, D. The modules above can be used to build a fault tree diagram, in any desired configuration, by PRISM’s module renaming feature.

4.5 Case Study

To show the applicability of our approach, we perform a quantitative analysis on the solar array case study, taken from [147]. We will present the obtained analytical results and a possible solution to improve reliability. Solar arrays are one of the most important components of any satellite mission, as they generate power for all other components of the satellite. The arrays are usually in a folded position during the launch phase of the satellite, becoming unfolded once the satellite is fully deployed in space. The goal of this component is to have the solar array aimed at the sun at all times in order to maximise power generation for the satellite. Fig. 11 shows the fault tree diagram of the mechanical components in the solar array. A detailed description of each component can be found in [147].

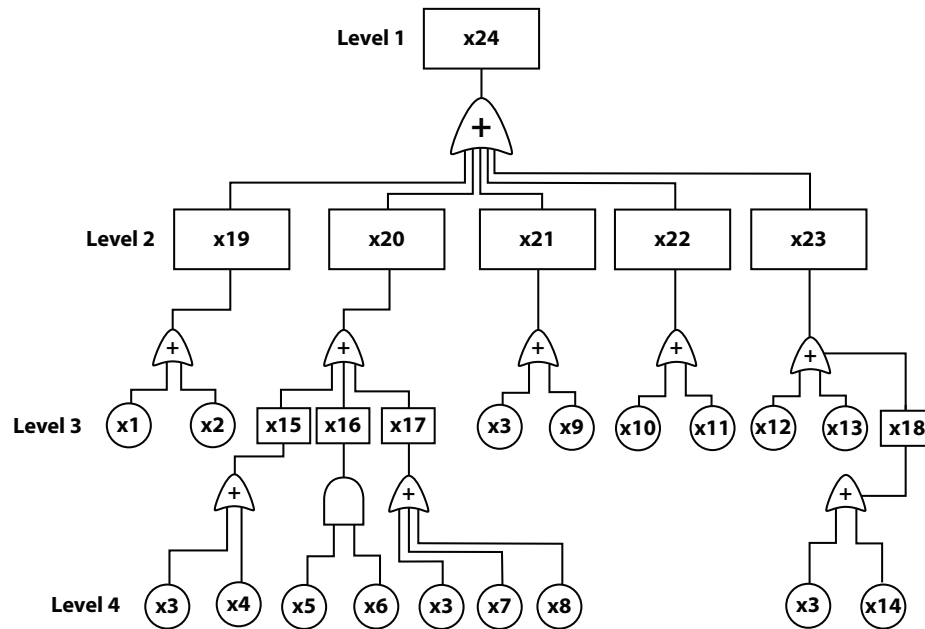


Figure 11: Solar Array Fault Tree

The top event X_{24} represents the failure of the solar array system. Intermediary events X_{19} , X_{20} , X_{21} , X_{22} and X_{23} , in the second level of the tree, represent the possible causes of failure in the solar array. At the third and fourth levels, X_1 , X_2 , X_3 , X_4 , X_5 , X_6 , X_7 , X_8 , X_9 , X_{10} , X_{11} , X_{12} , X_{13} , and X_{14} are the bottom events, that can cause a failure in events X_{19} , X_{20} , X_{21} , X_{22} and X_{23} . The solar array

fault tree was modeled first, as a DTMC and then, as an MDP. We reiterate that the choice of those types of Markov chain was motivated by the need to verify the system in a known and predictable environment as well as in an unknown environment. Every bottom event in the fault tree has a probability of being triggered and such a probability is based on the truth degree values specified in Table 3 [147]. For the sake of simplicity, we assume that only one bottom event can trigger at a time, with the exception of X_4 and X_5 , since those events are connected to an *AND* gate and thus both must be triggered to model the behavior of fault propagation. All other events in the tree are connected through *OR* gates. The objective of the experiment is to assess and compare the likelihood of faults originated at the different bottom events of the tree to reach the top event X_{24} causing a system failure. It is important to note that bottom event X_3 is connected to multiple gates in the system. This makes X_3 the most important bottom event in the tree since it has the highest probability of triggering a fault in another node. The model is encoded in PRISM following the approach specified in Section 5.3.1. The output of each logical gate has intrinsic probability of fault masking which, in the real world, means a non-destructive failure or a transient fault, which the system is able to detect and fix by itself, continuing its normal operation. The probabilities of fault masking are the same for all gates of the model. We performed our analysis using two different values for masking probabilities, 5% then 10%. However, since the model is parametric, any other value can be easily evaluated. The probabilities of each bottom event to reach a system failure are evaluated by verifying PCTL properties in PRISM, for both the DTMC and MDP models. The property for a node X_a , connected to an *OR* gate, is defined in PCTL as follows:

Property 1: $P_{max} =? [(F X_a = 1) \& (F X_{24} = 1)]$ - “What is the maximum probability that eventually X_a will trigger and eventually the fault will propagate to X_{24} , causing a system failure”.

The property for two nodes X_b and X_c , connected to an *AND* gate, is written as follows:

Property 2: $P_{max} =? [(F X_b = 1) \& (F X_c = 1) \& (F X_{24} = 1)]$ - “What is the

maximum probability that eventually X_b will trigger and eventually X_c will trigger and eventually the fault will propagate to X_{24} , causing a system failure”.

We use DTMC in the first experiment to assess the probability that a failure originated in a bottom event will reach the top event and the results are presented in Table 4.

Table 3: Fault Probability of Bottom Events

X1	X2	X3	X4	X5	X6	X7
4%	6%	10%	4%	6%	4%	8%
X8	X9	X10	X11	X12	X13	X14
6%	8%	3%	5%	8%	8%	8%

Table 4: Top Event Failure Probability(DTMC)

Event	5% Mask	10% Mask	Event	5% Mask	10% Mask
X1	0.0361%	0.0324%	X8	0.0514%	0.0437%
X2	0.0541%	0.0486%	X9	0.0722%	0.0648%
X3	0.0868%	0.0749%	X10	0.0270%	0.0243%
X4	0.0342%	0.0291%	X11	0.0451%	0.0405%
X5	0.0021%	0.0018%	X12	0.0722%	0.0648%
X6	0.0021%	0.0018%	X13	0.0722%	0.0648%
X7	0.0685%	0.0583%	X14	0.0685	0.0583%

The second set of experiments is conducted using MDP, to add non-determinism into the model. The scenarios are evaluated in the same manner as the DTMC experiment and Table 5 summarizes the result.

Table 5: Top Event Failure Probability(MDP)

Event	5% Mask	10% Mask	Event	5% Mask	10% Mask
X1	0.0361%	0.0324%	X8	0.0514%	0.0437%
X2	0.0541%	0.0486%	X9	0.0722%	0.0648%
X3	0.0902%	0.0810%	X10	0.0270%	0.0243%
X4	0.0342%	0.0291%	X11	0.0451%	0.0405%
X5	0.0041%	0.0034%	X12	0.0722%	0.0648%
X6	0.0041%	0.0034%	X13	0.0722%	0.0648%
X7	0.0685%	0.0583%	X14	0.0685	0.0583%

After comparing the results, it becomes clear that the increased probability of masking is considerably more effective for errors occurring in the lower layers of the fault tree.

It is notable that while most results are the same for DTMC and MDP, they do differ in $X3$, $X5$ and $X6$. The observed reason for these differences lies in the non-determinism present at the core of an MDP. *OR* gates are simple because one error is enough to trigger an output and a chance of propagation to the next node thus, according to our modeling assumptions, the path from bottom to top event becomes more linear and has less room for randomness. For an *AND* gate the scenario is different because the output generation and propagation depends on faults that occur

on two bottom events concurrently. As such, there are multiple ways the error scenario can play out (input 1 before input 2, input 2 before input 1, input 1 but not input 2 and so forth), thus opening more possibilities for non-determinism.

It is also of importance that event $X3$ is connected to multiple OR gates at the same time, allowing for non-determinism. Another very important observation is that the drop in system failure rate between 5% and 10% masking in the AND gates of the MDP model was 17.07%, which is the biggest gain in reliability observed in this experiment. Although the overall reliability of AND gates is greater in the DTMC model, the reliability gain of those gates is higher in the presence of non-determinism. Our analysis shows that the main causes of system failures in the solar array are components $X3$, $X9$, $X12$, and $X13$. Remarkably, the bottom events that generate all the above failures are located in the second layer of the fault tree and are connected through OR gates. As an additional experiment, we take the number one cause of system failure listed above (bottom event $X3$ propagating through OR gate to $X21$) and we run a test in a hypothetical scenario where we add system redundancy and replace the OR gate with an AND gate, as seen in Fig. 12. Since $X21$ and $X24$ are connected through an OR gate, we ignore all other connections to that gate.

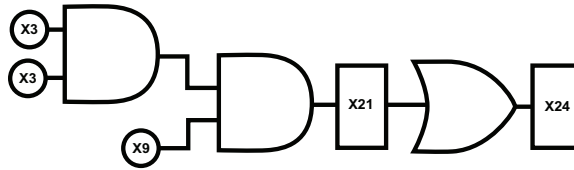


Figure 12: Redundancy Test

In such hypothetical scenario, the drop in system failure would be of 99.2%, using a DTMC model and a masking rate of 10%. At the end of these experiments, it is clear that the best way to improve the reliability of a fault tree, and consequently the system that the tree is based on, is to place the most critical nodes under AND gates, preferably with system redundancy, and to place such critical nodes further down in the tree.

4.6 Conclusion

The accuracy of the failure assessment is of utmost importance in safety-critical environments, where a system error may lead to catastrophic outcomes. In this paper, we have proposed a methodology for accurately performing fault tree analysis using

probabilistic model checking. The technique consists of modeling the logic gates in a fault tree diagram into DTMCs and MDPs that are encoded into the probabilistic model checker PRISM. The methodology is, then, used to conduct a probabilistic analysis on a solar array fault tree diagram, focusing on the likeness of a single fault to propagate and cause a top event failure. A comparison is shown between the results of both Markov chain models, with an analysis conducted on those results, pointing out the FTD's critical points. Building upon the methodology presented in this paper, a few other elements can be added to make for more complex FTA scenario, such as the introduction of time and concurrency of events.

Chapter 5

Article II: Efficient Probabilistic Fault Tree Analysis of Safety Critical Systems via Probabilistic Model Checking

Authors: Marwan Ammar, Ghaith Bany Hamad, Otmane Ait Mohamed, Yvon Savaria

Abstract: The cost and complexity involved in the development of critical systems encourage the use of reliability assessment techniques as early in the design cycle as possible. Existing techniques often lack the capacity to perform a comprehensive and exhaustive analysis on complex redundant architectures, leading to less than optimal risk evaluation. This paper addresses the aforesaid weakness by 1) proposing a new probabilistic modeling of Fault Tree gates and their composition as Markov Decision Process; 2) developing a new formal-based technique to perform an in-depth verification of the system's reliability. This technique makes use of the expressiveness of fault trees and the power of probabilistic model checking in order to investigate the best Triple Modular Redundancy partitioning and configuration of a system. The presented approach greatly improves the overall scalability with respect to other techniques, while also improving the accuracy of the results. For example, we can provide probabilistic failure rates for a chain of 100 redundant components in little

over 1 second.

5.1 Introduction

Ensuring the proper functionality of safety-critical systems is of utmost importance and, as such, dealing with faults at early stages of development has become increasingly important. To this end, many techniques for fault analysis and fault tolerance have been developed. In this context, Fault Trees (FTs) have gained widespread acceptance for quantitative safety analysis. FTs are top-down graphical representations of various combinations of lower level events that may cause the system to reach a top level failure (i.e., system failure). Fault Tree Analysis (FTA) can provide insightful information to designers regarding the reliability of their systems, such as how is their system most likely to fail and what are the most efficient ways to make it safer. Traditionally, FTA is performed by converting the system's FT into a Boolean function and simulating that function with different low level component failure rates [30–32, 35, 51, 124, 138]. However, these approaches are costly in time and resources. This is mainly due to the fact that their modeling of FT is limited to Boolean representation, which exponentially increases the resource requirements to reach the desired results.

Redundant architectures, such as Triple Modular Redundancy (TMR), are broadly used as alternatives for fault tolerance, in order to improve the reliability of safety-critical systems. In the literature, the utilization of formal methods to analyze these architectures is rather limited. In [?] the behavior of a single TMR is formalized through Communicating Sequential Processes (CSP). The work in [152] uses Uppaal model checker to analyze a timed automata model of a TMR system design. Both of these techniques are limited to a single system and do not consider multi-staged TMRs. In [33, 34], the fault tree of a redundant system is modeled and analyzed through Satisfiability Modulo Theories (SMTs), giving an estimation of the reliability gain across different TMR configurations. These approaches present new problems due to their intrinsic modeling approach, namely the amount of redundancy required to perform the analysis. Furthermore, their proposed modeling has the same limitation as previous approaches because they also rely on boolean representation.

Due to the increased cost and physical size constraints, the main challenge of implementing fault tolerance through TMRs is knowing where in the system to apply the redundancy and knowing which TMR configuration is the most beneficial to the overall reliability. In order to address the above mentioned issues, a new formal probabilistic FTA methodology is proposed. This work is distinct in the following ways: 1) A new probabilistic modeling of FTs is introduced. This approach consists of modeling the behavior of each FT gate as a probabilistic automaton (PA). Thereafter, a Markov Decision Process (MDP) model of the system’s FT is obtained by the parallel composition of all the PAs that compose the FT. 2) A recursive formal probabilistic analysis of the MDP model of the FT is proposed. This is achieved by utilizing the efficiency of Probabilistic Model Checking (PMC) techniques. In this analysis, a Component Contribution Investigation (CCI) is performed. The CCI consists in the evaluation of the contribution of the failure of each subcomponent to the system’s failure. 3) A new methodology to recursively evaluate the impact of TMR on the reliability of the system. The goal of this technique is to determine the best TMR partitioning and configuration for a system. The impact of the application of different TMR configurations on each critical subcomponent is investigated. This process is repeated until the resultant system reliability falls below the desired system failure rate.

The proposed methodology is experimentally evaluated on different system architectures, including the ones analyzed in [33, 34], demonstrating clear advantages. Our methodology is more scalable and provides approximately 80 times of speed-up. For example, the technique proposed in [33] fails to analyze a chain of 10 TMR components. The technique proposed in [34] is able to analyze a chain of 140 TMR components in 110 seconds, while our approach can analyze the same chain in less than 1.5 seconds. Our methodology significantly widens the possible analyses of TMR architectures. As well as evaluating the best TMR configuration (like in [33, 34]) our methodology can determine the optimal TMR partitioning and the impact of TMR at different FT levels.

The rest of this paper is structured as follows. In Section 5.2, the most relevant related works are discussed. In Section 5.3, we present the proposed methodology. In Section 5.4, we describe the experiments and report the main results. In Section 5.5, we draw some conclusions and discuss future work.

5.2 Related Work

Fault tree analysis (FTA) is a widely used method for risk assessment, mainly in the area of avionics, nuclear, and chemical industries [138]. FTA follows a deductive approach, which means it starts from an undesirable general event in order to find the origins of said event. In the context of FTA, the general event is known as *top level event* (TLE), from which the fault tree branches out vertically. The top event is defined as the failing point of a system in operating conditions, whether those conditions are considered normal or abnormal. Basic events are the ones at the very bottom of the fault tree, called *low level events*(LLE), independent from any other events. A quantitative FTA is performed by assigning fault probabilities to the bottom events which will dynamically be distributed through the rest of the tree. The relationship between fault tree events is expressed using relational constructs called *logic gates*, such as *AND* and the *OR* gates.

In [124, 125] quantitative and qualitative FTA techniques are proposed, respectively. These techniques consist of converting the FT into a format compatible with Shannon’s decomposition (i.e., boolean formulas). Thereafter, the proposed analysis is performed on the FT boolean formula to generate fault configurations that can cause system failure. The problem with these approaches lies in the boolean representation which limits their expressiveness of the system’s behavior. For instance, it is not possible to trace the source of the error without performing step-by-step comparison between the faulty and the reference models of the system. The works in [31] and [30] focus on the analysis of dynamic FTs. The FTs are represented as cause-effect diagrams known as *Bayesian networks* The analysis is performed through the Galileo tool [128], which performs FTA through the use of DIFTree algorithms.

A model-based safety assessment approach to analyze redundant components is proposed in [33]. In this technique, SMT solvers are utilized to generate all fault configurations that can lead to a system failure. This technique is an improvement (in terms of speed and accuracy) over traditional reliability analyses [70?], which utilize manual algorithms. However, the problem with the approach proposed in [33] lies in the amount of overhead it requires to generate the model. This is mainly due to the need to incorporate a reference and a redundant model of each subcomponent. Moreover, the redundant model of the subcomponents, must be duplicated, with one added multiplexer between each duplicated items, to allow fault injection. As

such, this modeling approach is very costly, with serious scalability issues due to the complexity and the size of the final SMT model. According to [33], with such modeling approach, even the most efficient SMT solvers cannot handle a chain of more than 10 TMR components.

An attempt to address the scalability issues of [33] is presented in [34], where the complexity is reduced through predicate abstraction. This technique replaces the reference and the redundant models with their abstracted versions to allow larger systems to be modeled. However, even if the behavior of the models is preserved, another problem arises with [33, 34]. Generating all possible fault configurations that may lead to a system failure is impractical. According to [33], a chain of 100 TMR components can have $2^{6 \times 100}$ possible fault configurations. An SMT solver can only generate a subset of these possible fault configurations (i.e. the obtained system failure rate will be underestimated).

5.3 Proposed Methodology

In this section, the proposed probabilistic fault tree analysis of redundant architectures is introduced. The flow chart of the proposed methodology is shown in Fig. 13. We start from a system-level model, in which a system is composed of interconnected hardware components. This system-level model must be provided by the designer in a general-purpose modeling language, such as SysML [61]. The failure rate of each component is characterized from the system-level specification. From the SysML model, a fault tree of the system is automatically synthesized using the tool proposed in [99]. Subsequently, a formal MDP model of the system’s FT is obtained through the parallel composition of the PAs of the FT gates. A library of the PAs of the FT gates is modeled a priori and each gate can be instantiated as needed. In this work, the composition of the MDP model of the FT is done automatically through the probabilistic model checker PRISM. The next step is to exhaustively analyze the generated MDP model in order to evaluate the maximum probability of the TLE (i.e., compute the system’s failure rate), in the presence of all LLEs. This is done by performing probabilistic model checking of the following property: “*What is the maximum probability that eventually TLE will be true (i.e. system will fail)*”. According to the specification and based on the criticality of the system, a threshold

for an acceptable failure rate is defined. If the computed failure rate is above the expected threshold, then an exhaustive analysis of the fault tree is required. In this case, we perform a Component Contribution Investigation (CCI) which is an examination of the impact of the failure of each subcomponent to the system failure. This is done by verifying the following property: *“What is the maximum probability that if component A eventually fails, the TLE will eventually fail”*

For each system, a Component Contribution Matrix (CCM) is created. The CCM is a library that records the impact of the failure of each subcomponent to the system’s failure probability. Subsequently, the proposed methodology investigates the effects of applying TMR on the critical components (based on the system’s CCM) of the system. If the system’s failure rate is above the desired threshold, the CCI is repeated and the CCM is updated. The previous steps are repeated until the system satisfies its specification. In the next subsections, we explain in detail the proposed modeling and analysis.

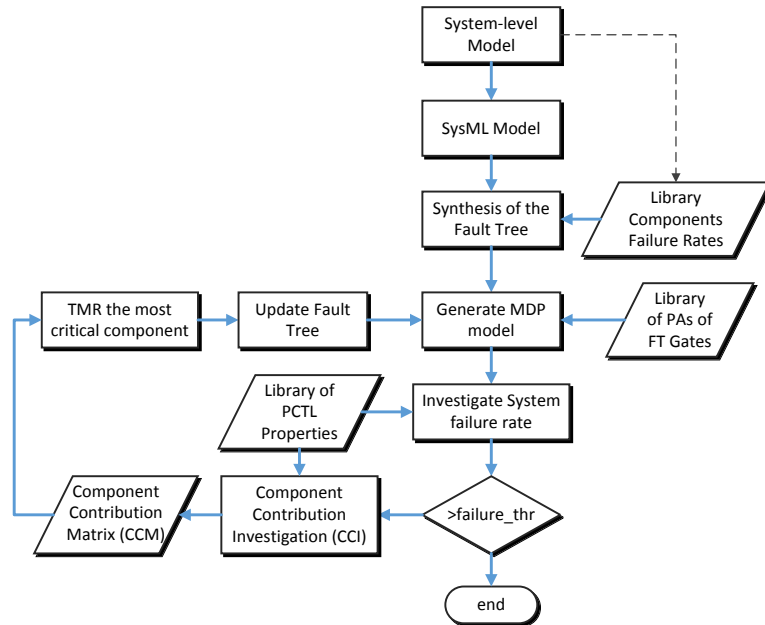


Figure 13: Main Steps of the Proposed Methodology

5.3.1 Probabilistic Modeling of Fault Trees

In fault trees, the set of events that might lead to a top level event failure is defined in a top-down manner and connected using FT gates. In order to accurately model the fault dependencies in FT, our approach focuses on the formalization and modeling of

the probabilistic behavior of FT gates. The general probabilistic model (automaton) of any FT gate is expressed in Definition 1.

Definition 1: Given a FT gate with a set of inputs X and an output Z , connected through a certain logic (such as AND, OR). The probabilistic automaton (PA) of this gate can be formally defined as a finite transition system as follows:

- S , a finite set of states;
- \bar{s} is the initial state $\bar{s} = S_0$;
- $\delta \subseteq (S \times S)$, a set of transitions between the states;
- Fr , a set of components failure rates;
- $\pi : \delta(S \times S) \rightarrow (\lambda, 1 - \lambda)$, a transition function assigning probability $\lambda \in Fr$ for each transition.

Example: Given two input events X and Y and their output Z , connected through an *AND* gate, output Z becomes true if and only if X and Y are true. Two examples of the PAs of one static and one dynamic gates are depicted in Fig. 14. For the static AND gate (see Fig. 14(a)), starting from an initial state $S_0(X=0, Y=0, Z=0)$ the next state can either be $S_1(X=1, Y=0, Z=0)$ or $S_2(X=0, Y=1, Z=0)$, with probabilities $p1$ and $p2$, respectively. At this point, the system can either move from S_1 or S_2 to $S_3(X=0, Y=0, Z=1)$, with probability $p2$ or $p1$, respectively, signifying fault propagation, or stay in S_1 or S_2 , with probabilities $1-p1$ or $1-p2$. A dynamic gate is modeled in the same manner, but with an added degree of complexity due to the priority constraint. A *Priority AND (PAND)* gate model (see Fig. 14(b)) starts at state $S_0(X=0, Y=0, Z=0)$. The model may move to $S_1(X=1, Y=0, Z=0)$ with probability $p1$, or to $S_2(X=0, Y=1, Z=0)$ with probability $p2$. If the model is at S_1 and the priority of X is higher than the priority of Y , and event Y happens, the next state will be $S_3(X=1, Y=1, Z=1)$, with probability $p2$. If the model is at S_1 and the priority of Y is higher than the priority of X , and event Y happens, then the next state will be S_2 , with probability $p2$. Lastly, if the model is at state S_2 and the priority of Y is higher than the priority of X , and event X happens, the model moves to S_3 , with probability $p1$. If the model is at S_2 and the priority of X is higher than the priority of Y , and event X happens, the model will move to state S_1 with probability $p1$.

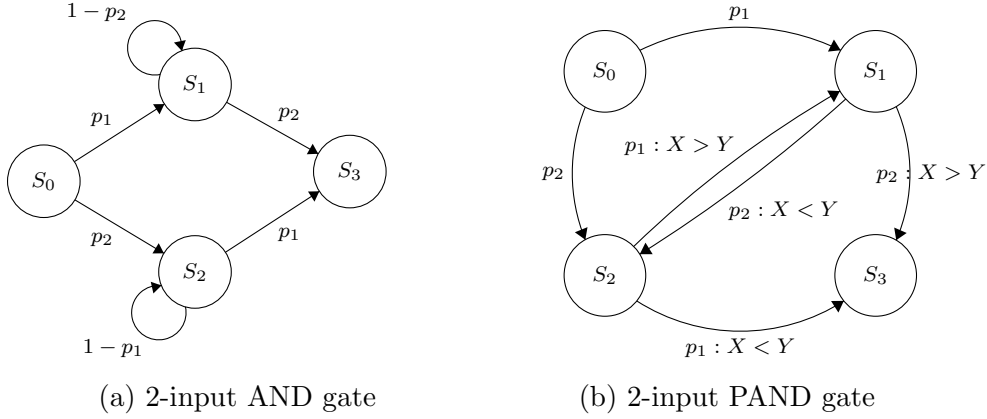


Figure 14: Example of Fault Tree Gate Automata

Due to space constrains, only the PAs of the *AND* and *PAND* gates are shown in Fig. 14. However, a list of all the FT gates that we have modeled and a brief description of their behavior can be seen in Table 6. After the PAs of the gates in the FT are modeled, the MDP of the FT is constructed. In this work, the MDP definition developed in [90] is adapted. An MDP is defined as a finite transition system $(S, \bar{s}, A, P, L, \mathbf{Steps})$ as follows:

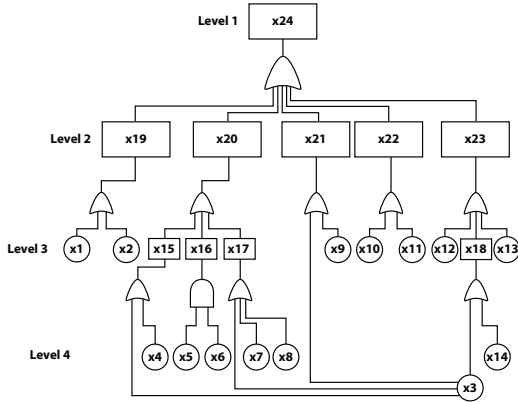
- S , a finite set of states;
- \bar{s} is the initial state $\bar{s} = S_0$;
- A is a set of actions (i.e., LLEs);
- AP is a set of atomic propositions;
- $L : S \rightarrow 2^{AP}$ is a labelling function that provides, to each state $s \in S$, a set $L(s)$ of atomic propositions;
- $\mathbf{Steps} : S \times Act \rightarrow Dist(S)$ is the (partial) transition probability function, with $Dist(S)$ denoting the set of all discrete probability distributions over S .

The MDP, as defined previously, is a stochastic system where all the decisions are made in a non-deterministic manner. At each state, a non-deterministic choice is done from a finite set of possible actions by parallel synchronization (\parallel_S) of the PAs ($M_{MDP} = \{PA_1 \parallel_S PA_2 \parallel_S \dots \parallel_S PA_n\}$). Subsequently, the function \mathbf{Steps} randomly chooses the successor state according to the probability distribution $\mathbf{Steps}(s, a)$.

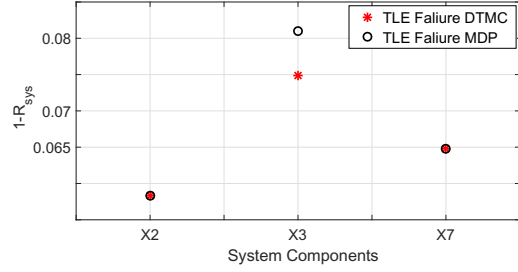
The choice of MDP over other types of Markov models, such as the Discrete-Time Markov Chain (DTMC), lies in the fact that real life systems experience nondeterministic behavior due to the concurrency between components operating in parallel. Concurrency can be found when an event leads to several different errors (i.e., impacts several subcomponents) in the system. A DTMC model, in this case, divides the probability of failure equally over all the affected subcomponents. However, this behavior is not realistic. The MDP implementation of the aforementioned event will result in a completely random distribution choice, thus reflecting more accurately a real environment. An example of this can be seen in Fig. 15, where the effects of non-deterministic (i.e., MDP) versus probabilistic (i.e., DTMC) distribution on the TLE failure rate ($1 - R_{sys}$) of the solar array system of the DHF-3 satellite [147] is investigated. The solar array FT can be seen in Fig. 15(a). It can be observed from the results (shown in 15(b)) that the choice of model (MDP or DTMC) does not influence the computed failure rates when the events only affect a single subcomponent (e.g., $X2$ and $X7$). On the other hand, an event that affects several subcomponents (e.g., event $X3$ directly affects subcomponents $X15$, $X17$, $X18$, and $X21$) will have its impact on the system’s failure underestimated, in the DTMC model. It is important to mention that our probabilistic modeling approach takes full advantage of the concept of FT modularity [138], which means that a fault tree of a big system can be broken down into smaller sub trees. These smaller fault trees can then be analyzed separately, and the result of this analysis is fed back to the original FT.

Table 6: Modeled FT gates

Gate	OutputCondition
AND Gate	All inputs are true
OR Gate	At least one input is true
Inhibit Gate	In the presence of a trigger event, one other input must become true
Functional Dependency	When the trigger event occurs, the dependent basic events are forced to occur
Combination Gate	At least n inputs are true
Exclusive OR Gate	Exactly one input is true
Priority AND (Sequence Enforcing) Gate	The inputs must become true in a specific sequence
Cold Spare Gate	The primary input becomes true, followed by the cold spare inputs in the correct sequence



(a) Solar Array Fault Tree



(b) MDP versus DTMC

Figure 15: Modeling FT as MDP versus DTMC

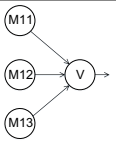
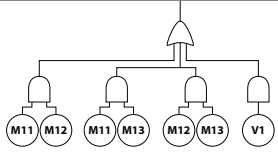
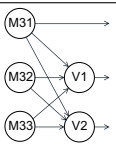
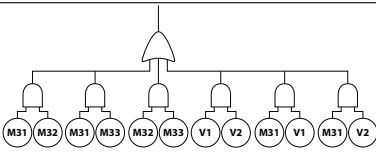
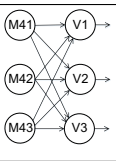
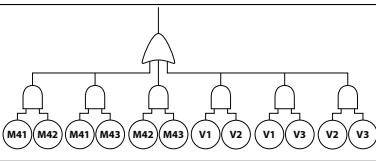
5.3.2 Modeling of TMR

The assessment of the impact of the optimal redundant architecture configuration is of great importance to the development of safety critical systems. Redundancy is commonly applied to components deemed essential to the system’s correct functionality. TMR is a broadly diffused architectural pattern for component redundancy [4, 54, 95, 129], which consists in triplicating a critical component and feeding each copy’s output to a majority voter. The voter evaluates each component’s output and returns the value computed by the majority. Applying TMR to the whole system is very costly and might not be required based on the system’s criticality level. Moreover, the efficiency of a TMR is dependent on the reliability of each component and TMR voter, as well as on the TMR configuration.

The proposed methodology determines the optimal TMR partitioning (where the TMR is required in the system) and the best TMR configuration to be used. Starting from identifying the best partition, the most critical component, according to the system’s CCM, is singled out. Following this, the system’s failure rate is evaluated for each TMR configuration. This process is repeated for all critical components until the desired system failure rate is achieved, as shown in Fig. 13.

Through our modeling, component redundancy can be applied in two ways: 1) the sub FT, of which the critical component is the TLE, is triplicated within the system FT and the new analysis is performed. 2) the sub FT of the critical component is extracted from the system FT, triplicated, and analyzed separately. The numerical results of the analysis are then fed back to the system FT, as *transfer in* events. The

Table 7: TMR configurations and Respective FTs

a		
b		
c		

first way involves increasing the size of the MDP model and, consequentially, the state space. However, the second way reduces the size of the model and increases the efficiency of the verification, while retaining the accuracy. This is achievable through FT modularity [138], combined with the proposed probabilistic modeling, which allows FTs to be divided into sub FTs without losing any properties. Therefore, throughout this work, the second method is applied.

In this work, all TMR configurations were analyzed, but for space constraints we only present the most commonly used ones. Table 7 shows the main three TMR configurations and their equivalent fault trees. The 1-voter and 2-voters TMR configurations can have different sub-configurations depending on how these TMRs feed their output to the next stage. For instance, a 1-voter TMR can have its output driven by only the voter, or by a combination of the voter output and the copies of the component. The number of outputs of a TMR and their origin are taken into account when building the equivalent FTs. As can be seen in the 2-voters TMR example, element $M31$ has more impact to the TLE than elements $M32$ and $M33$, due to the fact that it is the singled-out element. Another modeling option that impacts the analysis of redundant systems is the uniformity of failure rates of the TMR copies of the component. Failure rates can be uniform or non-uniform, meaning that the triplicated components can have either the same or different failure rates. The same principle applies to TMR arrangement, where a 2-voters TMR chain can feature homogeneous or non-homogeneous arrangement of voters at different stages, as seen in Fig. 16. The variation in the system reliability due to all the above mentioned TMR

design options is discussed in detail in Section 5.4.

5.4 Experimental Results

The proposed methodology is fully automated on the top of the well known probabilistic model checker PRISM [91], a free, open source probabilistic symbolic model checker, based on the reactive modules formalism [7]. The probabilistic automata of the fault tree gates (similar to Fig. 14) are modeled as PRISM modules. Each module is composed of a set of commands. A PRISM command is a tuple $cmd = (act, guard, rate, action)$ following the format $[<act>] <guard> \rightarrow <rate>: <action>$, where:

- act is an action label used for synchronization of the different levels of an FT;
- $guard$ is a predicate over the inputs of the FT gates;
- $rate$ is the probability of occurrence of a FT event;
- $action$ is a set of n updates that will translate into transitions in the FT.

Afterwards, an MDP model of the system is automatically generated by PRISM. The failure rate of the system is investigated by verifying a set of Probabilistic Computation Tree Logic (PCTL) properties over the MDP model. The maximum probability of a system failure can be obtained by verifying the following property:

$$\mathbf{PCTL\ 1 : } P_{max} =? [(F\ TLE = 1)]$$

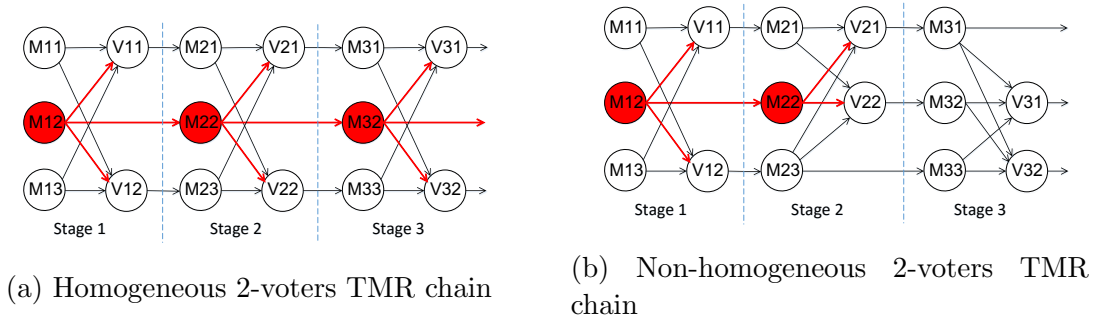


Figure 16: Example of different TMR arrangements

The contribution of the failure of a component to the system failure (i.e., CCI) is obtained by verifying the following property:

$$\mathbf{PCTL\ 2} : P_{max} =? [(F\ comp = 1) \& (F\ TLE = 1)]$$

The experiments have been performed on a machine with an Intel Core I5-4200U CPU and 8 GB of RAM. To show the applicability of our approach, different sizes of TMR chains are analyzed. This experiment is done with chains of n length, with 1, 2 and 3 voters, considering different TMR combinations. The length of the chain varies from 10 to 100 components. Fig. 17(a) shows the CPU time consumed to construct the MDP model and to perform the proposed analysis. Due to the efficiency of the proposed modeling, we were able to analyze a chain of 100 TMR components in little over 1 second. Fig. 17(b) brings the memory consumption for chains of different sizes. Fig. 17(c) depicts the relationship between the total number of states in the model and the length of the chain. As evidenced by the results, the resource consumption is small and grows linearly with the size of the chain. For example, a chain of 100 components is verified in less than 1 second and consumes less than 0.02 MB of memory. This is achieved through our efficient modeling approach, which virtually eliminates the problem of state explosion [132].

The second experiment consists in investigating the impact of the TMR configurations on system reliability. The analyses are performed on networks of 10 components (after applying TMR) connected in series, where we vary the failure rate of components and voters. In this experiment, we consider a uniform failure distribution (i.e. the triplicated version of each component have the same failure rate). Moreover, it is assumed that all the components feature the same TMR configuration. The results, depicted in Fig. 18, lead to the following observations: 1) when the failure rate of the voter ($1 - R_v$) is high and the failure rate of the component ($1 - R_m$) is low (see Fig. 18 (c,d)), the best TMR configuration is the one which employs two voters. This is mainly because that configuration reduces the impact of the failure of the voters by taking into consideration the low failure rate of the components. 2) When the failure rate of the voter is low and the failure rate of the components is high (see Fig. 18 (a)), the best TMR configuration is one voter, as the reliability provided is very similar to the other configurations but without the area and power overhead resultant from voter redundancy. From Fig. 18 (a,b) we can conclude that when the components

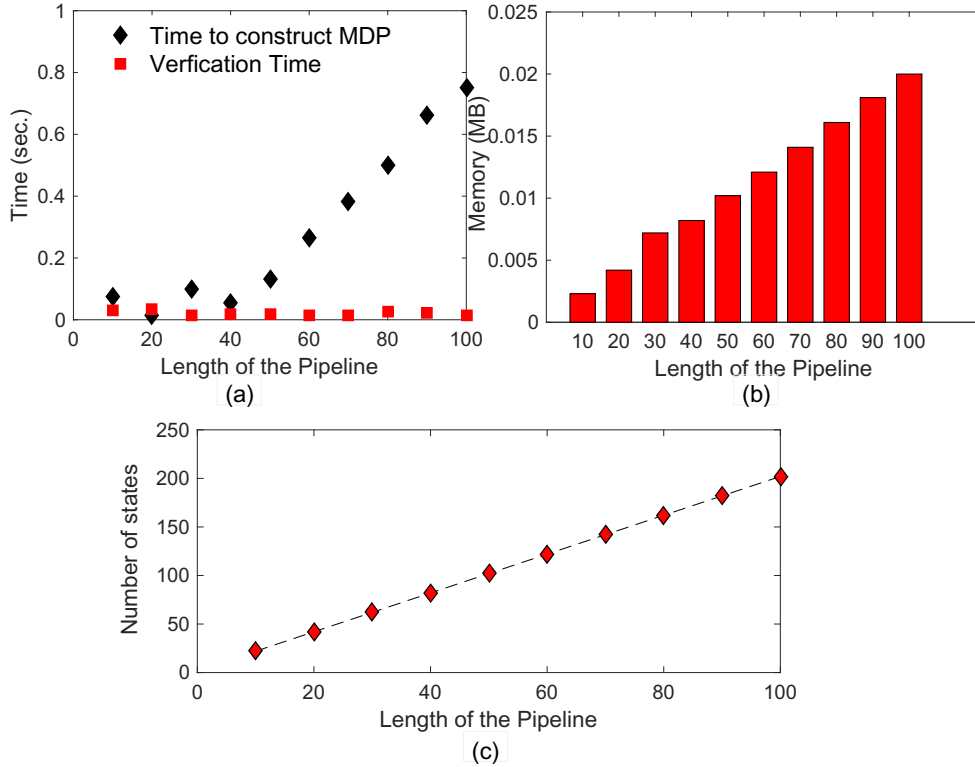


Figure 17: Results of Investigating the Proposed Methodology's Scalability

have high failure rates, the performance of two and three voters configurations is similar, and they outperform one voter configuration as the failure rate of the voter increases.

We further investigate TMR chains, considering non-uniform failure distributions (i.e., the triplicated components have different failure rates). To perform this analysis, it is assumed that the triplicated components are physically separated, thus they may be affected differently by environmental conditions (i.e. extreme heat, radiation, etc.). As such, we randomly increase the failure rate of one component (referred to as critical component) within each TMR.

Through the analysis of Fig. 19 (TMR3v) it is evidenced that the TMR with three voters is the best choice for nearly all situations. This happens because all components are “shielded” by the voters at all stages of the chain, thus the occurrence of a component failure is always masked. Three-voter TMRs are, however, the most costly option in terms of area and power. From the results of our analysis of TMRs of two voters, depicted in Fig. 19 (TMR2v), it can be observed that the homogeneous chain of two-voter TMRs is particularly sensitive to non-uniform failure conditions,

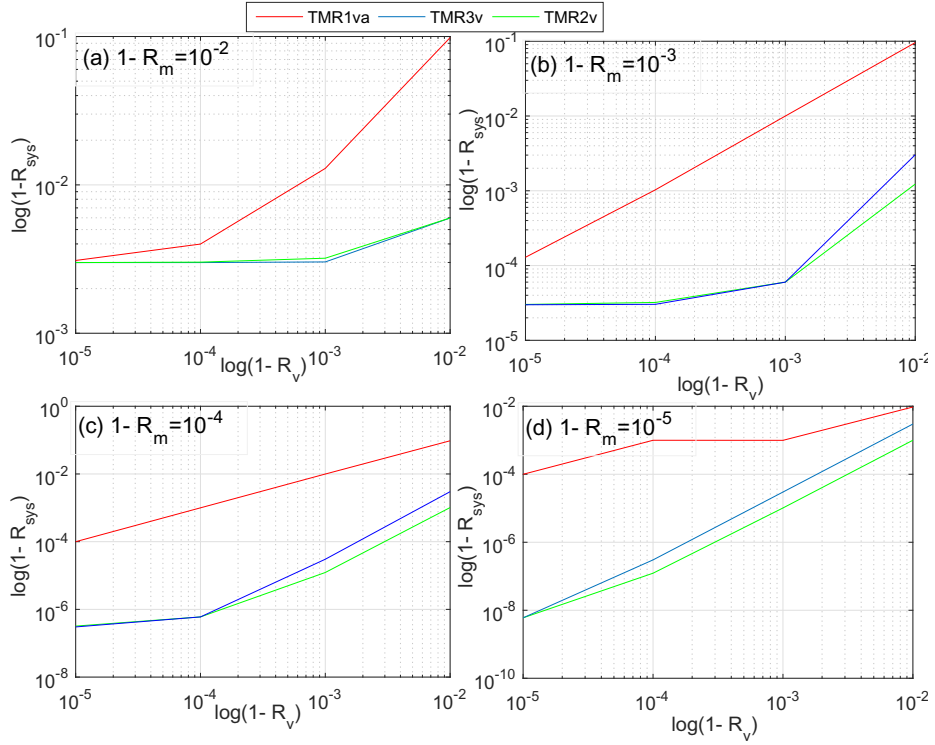


Figure 18: TMR chain with uniform failure rates

especially if the critical component is the singled out one. This is due to the fact that a failure in the singled out component feeds directly into the next singled out component until it reaches the system outputs and cause the system failure. This phenomenon can be seen in Fig. 16(a), where the failure of $M12$ carries over to $M22$, and subsequently to $M32$, and so on. Our results demonstrate that this weakness can be mitigated with the use of non-homogeneous chains of two-voter TMRs. As seen in Fig. 16(b), the configuration of the voters is changed from stage to stage. A fault originated in $M12$ propagates to $M22$ but it is masked by the voters $V21$ and $V22$. This is evidenced by Fig. 19 (TMR2v2), where significant improvement can be seen when compared to homogeneous chains.

Lastly, we perform analyses on the HSCOM subsystem of the HERMES Cubesat Satellite, responsible for primary high speed communications [28] and on the solar array system of the DFH-3 satellite. The fault tree of the HSCOM subsystem is shown in Fig. 20. Due to the lack of exact component failure rates in the design specification, all bottom events are assigned the same failure rate of 5.0×10^{-3} . *Transfer in* events (X15 and X19) have probabilities of failure equal to 1×10^{-2} each. Subsequently, a CCI is performed on the model and the results are reported in the CCM.

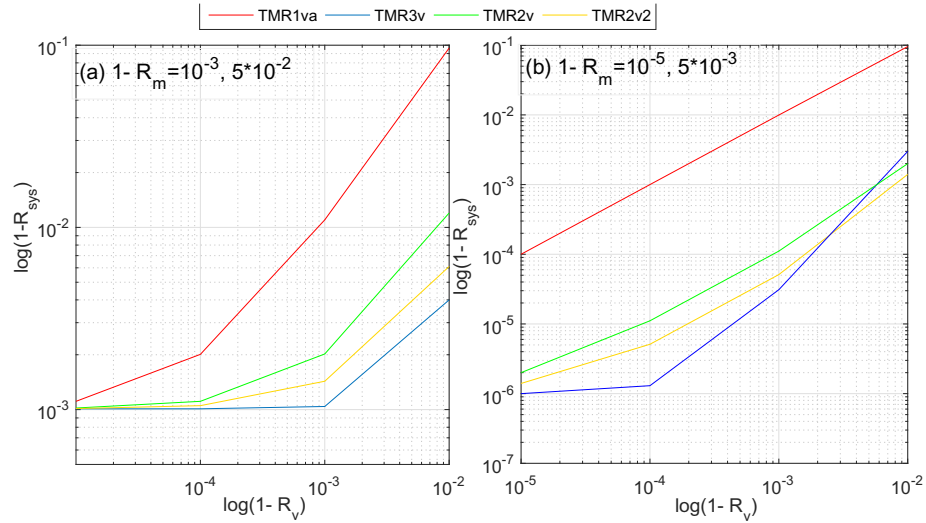


Figure 19: Variation in the system failure rate due to non-uniform distribution

Fig. 21(a) presents a comparison of the impact of TMR (1 and 3 voters) on the different components of the system. It can be observed that the most critical component in the system is $X16$, so we further investigate it. $X16$ is the top event of a sub FT composed by gates B and A . We will call the sub fault tree as BA . Upon further inspection on BA , it was observed that the most critical component in BA is $X8$. Thus a TMR is applied to $X8$. Fig. 21(b) contrasts the original system failure rate (no TMR) with the updated failure rate after applying TMR to each component in the system. To show the versatility of the approach, a similar set of experiments is performed on the solar array FT of the DFH-3 satellite, shown in Fig. 15(a). The solar array experiment follows the same steps described in the HSCOM experiment, with the difference that we use the actual failure rates of the FT components, obtained from [147]. A comparative result of the effectiveness of TMR on each component of the solar array is presented in Fig. 22(a). The impact of applying TMR on each component to the failure rate of the system is depicted in Fig. 22(b). It can be observed through the analysis that the impact of TMR on a multi-level structure FT is not straightforward, when compared to linear structures (chains of TMRs). This is mainly because the impact is not only dependant on the failure rates of components and voters, but also on the structure of the FT. This means that the same TMR configurations may have different levels of impact when applied to different FTs of different systems.

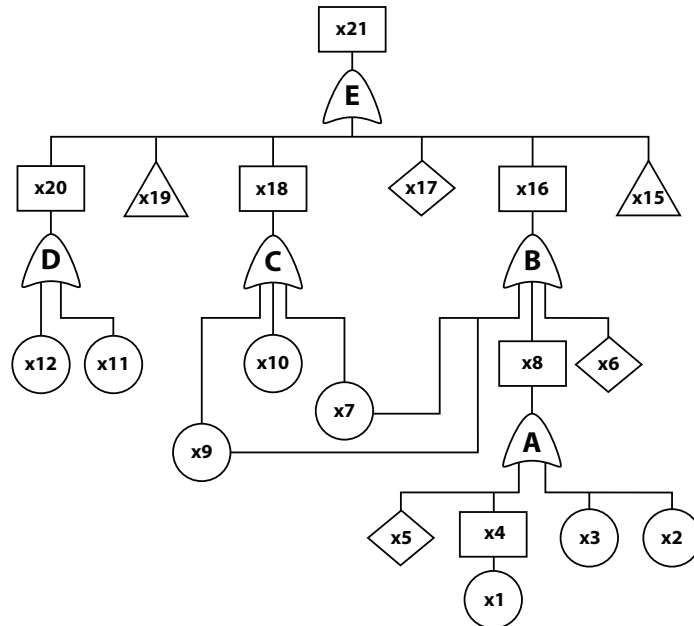


Figure 20: Hermes Cubesat HSCOM Fault Tree

Through our analysis, it is also observable that within the same sub tree, tripling lower level events is more efficient, compared to higher level events. This can also be seen in Fig. 21(b), where the impact of applying TMR on $X8$ or on $X16$ is practically the same. This is very insightful, as the area and power overhead resulting from tripling $X8$ is much smaller than the overhead generated from tripling $X16$ (see Fig. 20).

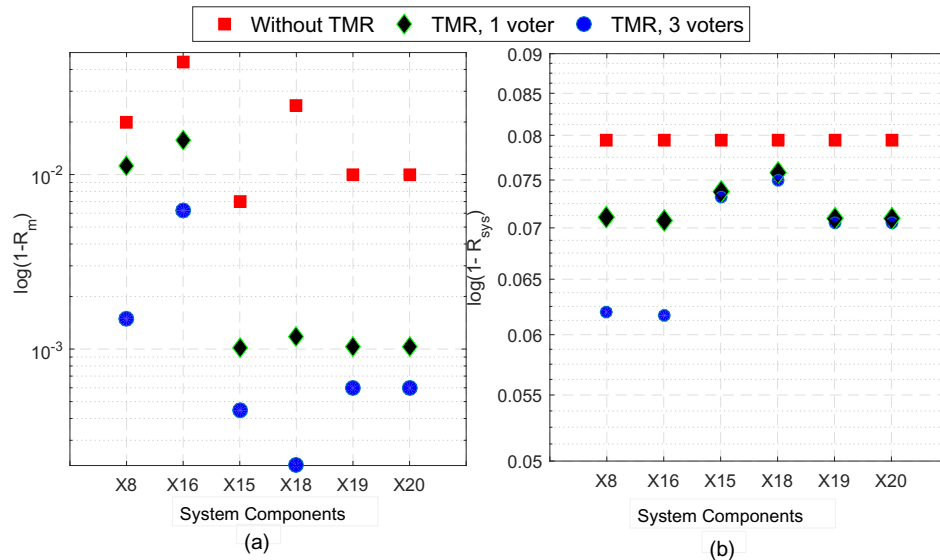


Figure 21: Hermes HSCOM Results

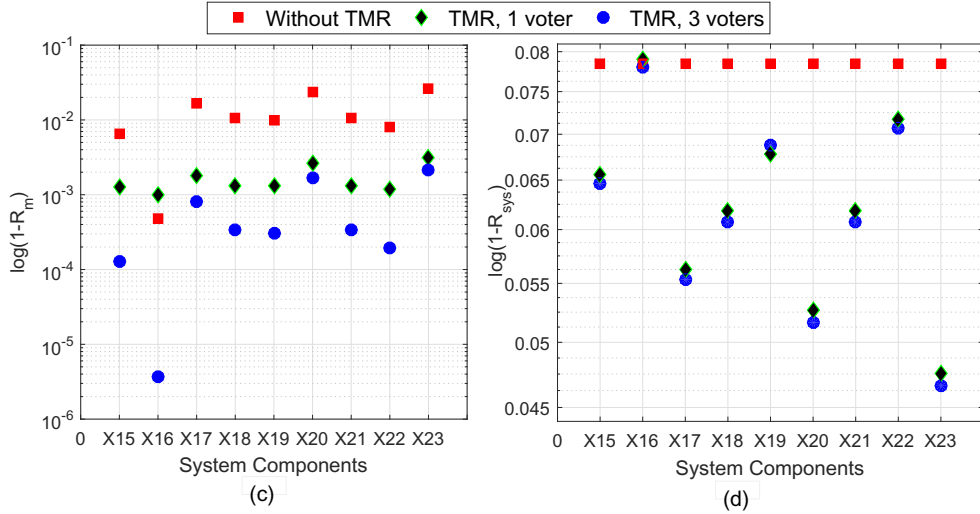


Figure 22: Solar Array TMR Results

Throughout the experiments it is observed that although the implementation of TMR usually impacts the system positively, the choice of the best TMR configuration is heavily dependant on the characteristics of the system and on the environment where the system is deployed. Fig. 18 shows that, in controlled environments (i.e., the triplicated versions of the component have the same failure rates), two-voters TMR provides the best reliability. In unknown environments that might influence the system in unpredictable ways (i.e., the triplicated versions of the component have different failure rates), the three voters TMR configuration is always the best choice, assuming that the failure rate of the voters is always smaller than the failure rate of the components.

5.5 Conclusion

In this paper, an efficient probabilistic modeling and analysis of fault trees is proposed. A new probabilistic approach for modeling fault trees was developed. These model as used to construct the MDP model of system fault tree. Next, the efficient PMC (i.e., PRISM) is utilized to perform the proposed exhaustive analysis. The proposed technique is extended to perform a recursive probabilistic analysis of TMR architectures. This technique is used to investigate the best TMR partitioning, configurations, and the impact of implementing TMR at the different levels of the system's FT. Our results demonstrate that the proposed methodology has great potential as

it is more scalable, orders of magnitude faster, and it provides better quality results when compared with contemporary techniques [33, 34]. For instance, the proposed technique is able to analyze a chain of 140 TMR components in less than 1.5 seconds. Also, our proposed methodology provides a more accurate system failure rate estimation in the presence of TMRs in contrast to other techniques ([33, 34]) can only investigate a subset of all possible fault configurations of large systems.

Chapter 6

Article III: System-Level Analysis of the Vulnerability of Processors Exposed to Single-Event Upsets via Probabilistic Model Checking

Authors: Marwan Ammar, Ghaith Bany Hamad, Otmane Ait Mohamed, Yvon Savaria

Abstract: Due to current technology scaling trends, digital designs are becoming strongly susceptible to space radiation effects. These effects can cause unwanted single event upsets (SEUs) in any state element. This paper presents a new system level model of SEUs propagation through processors as a Continuous-Time Markov Chain (CTMC). Moreover, probabilistic formal techniques (such as probabilistic model checking) are utilized to exhaustively estimate the impact of SEUs on the system behavior. The proposed CTMC model was analyzed for different SEU injection scenarios and different bit-flip rates. Results demonstrate that the proposed approach can provide an accurate estimation of different reliability metrics, such as Mean Time to Failure (MTTF), Mean Time to Recover (MTTR), and the probability of failure for each SEU injection scenario in the system's subcomponents. Furthermore, the proposed probabilistic system-level analysis was utilized to investigate the optimal self-repair rate required in the system to obtain the desired level of reliability.

Results demonstrate that in comparison with existing simulation techniques for fault impact evaluation, the presented approach can provide consistent results while being orders of magnitude faster in terms of CPU time.

Index Terms: Single event upsets (SEUs), system-level analysis, probabilistic model checking (PMC), PRISM model checker, continuous-time Markov chain (CTMC), mean time to failure (MTTF), mean time to recover (MTTR).

6.1 Introduction

With advances in technology, micro-electronic systems are becoming more vulnerable to soft errors induced by Single Event Upsets (SEUs). An SEU is defined as a change in the state of one or more memory elements inside a system [116]. This change in state can often be harmlessly fixed if detected by the system. However, the non-detection of such event may hinder the system in some cases, which may lead to critical consequences in safety-critical applications, such as space missions and avionics.

The traditional and most direct approach to evaluate the SEU vulnerability of a system (i.e., an application running in a processor) is through a process called *dynamic radiation ground testing* [26, 134]. This method consists in exposing the target system to a radiation flux and counting the number of errors observed. The outcome is computed in the form of a parameter known as the *dynamic cross-section* (σ), which is defined as the ratio between the number of errors observed at the output of a Design Under Test (DUT), divided by the fluence of hitting particles [116]. A problem with that metric is that any change in the application requires a new dynamic test, thus resulting in an expensive and time-consuming method.

Alternative methods for SEU estimation have emerged, with the goal of reducing the time and cost constraints associated with *dynamic radiation ground testing*. In [117, 135], the authors introduce a method of injecting SEUs at random time intervals through emulation, by making use of an interrupt routine to alter values within the processor's internal registers and memory. Fault injection through emulation is also used in the *direct memory access SEU emulation* method [55], where a dedicated hardware component, controlled externally, selects the time instant and the bit to be altered in the memory. This approach is further explored in [56, 57], where the SEU

injection is performed through probabilistic models of the system, with the goal of estimating the system’s time to failure (TTF) and time to recover (TTR). However, this technique still requires emulation in order to obtain certain system rates which the model is built upon (i.e., coverage factor, error factor, and failure factor). Another branch of SEU estimation techniques focus on fault injection through simulation, which is usually done by injecting faults at logical or electrical levels [64, 83, 85]. The advantage of these techniques is the high level of control over the fault injection scenarios, since the user has free access to the entirety of the system and the timing of the injections is very accurate. However, emulation and simulation based techniques have severe drawbacks. Disregarding the considerable time required to simulate or to emulate a scenario of thousands of injected faults [17], both approaches are limited in terms of accuracy. This problem arises due to the fact that these techniques are not exhaustive, but rather reliant on input vectors [84]. In simulation-based verification, the mind-set is first to generate input vectors and then to derive reference outputs. Simulating a vector can be seen as verification through input space sampling. This means that unless all points are sampled, there exists a possibility that an error escapes verification.

Recently, the use of *formal based techniques* to analyze soft errors at logical and higher abstraction levels has been proposed, such as the work done in [23]. These techniques provide new insights into the vulnerability of digital designs to SEUs. This is mainly because they are exhaustive and not limited by the number of test vectors as in simulation based techniques. In formal techniques, the user starts out by stating what output behavior is desirable and then lets the formal checker prove or disprove it. In other words, given a property, formal verification exhaustively searches all possible input and state conditions for failures. However, at logical abstraction level, these techniques suffer from *state explosion* (i.e., exponential growth in the number of states of the model) [65]. Therefore, it is expected for these techniques to be more efficient at higher abstraction levels, such as system-level.

In this paper, a new approach to compute an accurate estimate of a system’s vulnerability to soft errors is introduced. The propagation of SEUs is modeled as a Continuous-Time Markov Chain (CTMC) based on probabilistic model proposed in [57]. The analysis of the obtained model is performed using Probabilistic Model Checking (PMC) [19]. PMC is a fully automatic and exhaustive technique that has

been successfully employed in a large scope of application domains, such as communication and multimedia protocols, security and power management. The proposed system-level approach focuses on modeling the system details related to the SEU propagation path through the processor rather than its logical behavior. Subsequently, the analysis performed consists in the probabilistic evaluation of the Mean Time to Failure (MTTF), the Mean Time To Recover (MTTR), and the expected system availability in different SEU injection scenarios. The obtained results are compared with values reported in [56]. Additionally, the case study reported in [118] is also modeled using the proposed approach. The analysis consists in the computation of the contribution of each component of the system to a failure. The obtained results demonstrate that comparing with existing simulation techniques, the proposed approach provides consistent results while being orders of magnitude faster. Moreover, the approach is very versatile, allowing the execution of customizable tests, where the user can stipulate the inputs and the expected outcome, and the model automatically investigates the results.

The rest of the paper is organized as follows. Section 6.2 discusses the steps taken to generate the proposed Markov model of SEU propagation over time. Section 7.4 offers an overview of the adopted probabilistic modeling for the instruction cycle of processors (fetch, decode, and execute), including the self-repair scenario. In Section 6.4, the implementation of the proposed CTMC model in PRISM is introduced. Section 10.2.6, explains the experiments and results obtained, and Section 10.2.7 concludes by summarizing the main contributions of this work and the results.

6.2 Markov Modeling of Self-Repair Systems

In this subsection, we utilize the model introduced in [9], where the behavior of a fault-tolerant micro-electronic system subjected to the effects of ionizing radiation is defined. This model is adapted from the one proposed in [57]. At a certain time t , a system exposed to SEUs can operate in one of three states (F_0 , F_1 , and F_2), as shown in Fig. 23 [9]. The probability that the system is in state F_i at time t is denoted by $P_{F_i}(t)$. The system is said to be in state F_0 when operating error-free. When an SEU triggers an error which is promptly detected, the system moves to state F_1 . The system remains in this state for the amount of time required to recover from the error, usually through a reset signal that restarts the control unit and restores correct

functionality. After being restored, the system moves back to state F_0 . In the case of an error that is not detected within the time limit, the system moves to state F_2 . It will remain in this state until the error is detected, at which point the system moves to state F_1 , and finally back to state F_0 . As such, the probability distribution of future states of a system exposed to SEUs depends only upon the present state and not on the sequence of events that preceded it, which characterizes a Markov model.

The time interval in which the system stays in state F_0 before the occurrence of an error is known as time to failure (TTF). Similarly, the amount of time spent in state F_2 before eventually going back to state F_0 is called time to recover (TTR). Since a system in state F_1 immediately detects the error and performs the restoration cycle, the amount of time spent in that state is negligible when compared to the time spent in other states, thus the time interval in which the system is in state F_1 is ignored in the remainder of this paper.

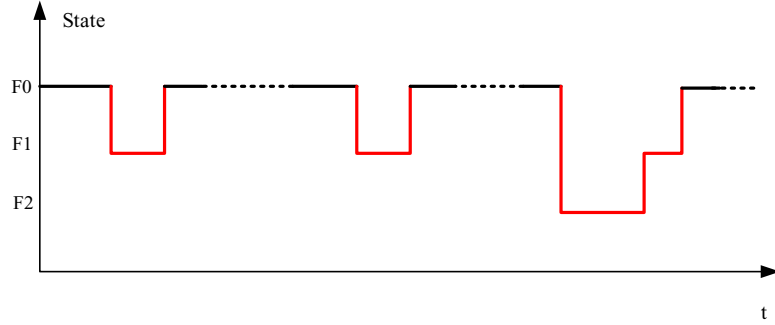


Figure 23: Time Progression of a Fault-Tolerant Micro-Electronic System Exposed to SEUs [9]

Based on the discussion above, the probability distribution of $P_{F_0}(t)$ and $P_{F_2}(t)$ can be obtained. Starting with $P_{F_0}(0) = 1$ and $P_{F_2}(0) = 0$ (i.e., no errors at $t=0$), the probability of the system being at P_{F_0} at time $t + \delta t$ (see Eq. (1)) is given by the addition of two mutually exclusive probabilities: 1) The system is in state F_0 and the SEU is detected. 2) The system is in F_2 and the error is detected after the time limit. The probability of the system being at P_{F_2} at time $t + \delta t$ (see Eq. (2)) is given by the addition of two mutually exclusive probabilities: 1) The system is in state F_2 and the error is not detected. 2) The system is in state F_0 and the SEU is not detected within the time limit. In Eq. (1) and (2), P_D is the probability that the SEU is detected, P_{ND} is the probability that the SEU is not detected, and δt is a

time increment.

$$P_{F0}(t + \delta t) = P_{F0}(t) \cdot P_D(\delta t) + P_{F2}(t) \cdot P_D(\delta t) \quad (1)$$

$$P_{F2}(t + \delta t) = P_{F2}(t) \cdot P_{ND}(\delta t) + P_{F0}(t) \cdot P_{ND}(\delta t) \quad (2)$$

The probability of an error being detected by the system is given by Eq. (3), where N_{inj} is the total number of SEUs injected, N_{det} is the number of SEU that can be detected by the system, and T is the amount of time in which the system is exposed to SEUs.

$$P_D(\delta t) = 1 - \left(\frac{N_{inj}}{T} - \frac{N_{det}}{T} \right) \delta t \quad (3)$$

Next in this paper, the concepts discussed here are used to formalize a self-repair system as a Markov model, and to perform a study of the impact of SEU on the behavior of processors through model checking.

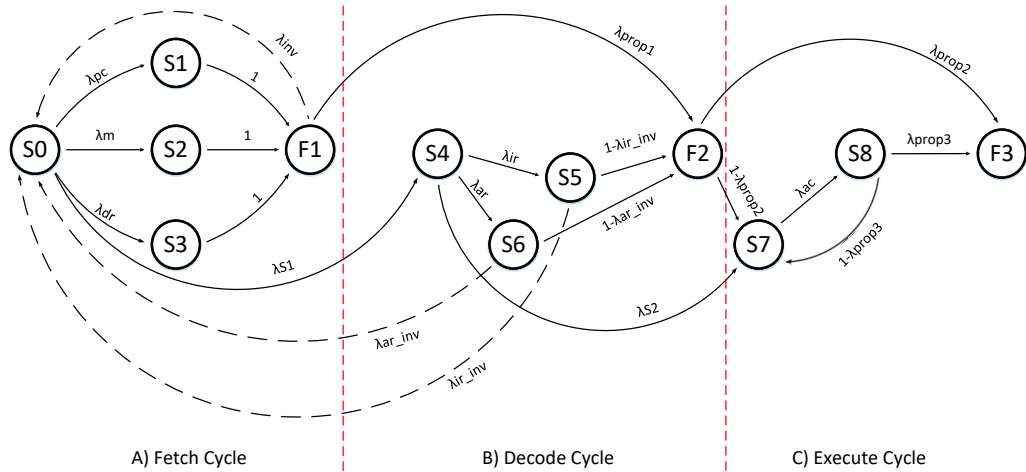


Figure 24: Proposed Probabilistic Model of SEU Propagation Through a Processor Instruction Cycle.

6.3 Proposed Probabilistic Modeling of SEUs Propagation in Processors

The execution of each instruction follows an *instruction cycle* composed of three phases: the *fetch* and *decode* phases, which are handled by the control path, and the *execute* phase, which is handled by the data path [37]. This instruction cycle is carried by the micro-operations that are performed during each of the different phases. This work considers a processor design that can access N bytes of memory.

Each instruction is a word of length n bits, consisting of an operation code of length n_{op} , and an address of length n_{adr} , where $n = n_{op} + n_{adr}$. This processor has a programmer-accessible register, labelled *accumulator* (AC), of size n . In addition to AC , this processor has other registers needed to perform the internal operations *fetch*, *decode*, and *execute*. The registers are the following:

- Data Register (DR): n -bit register which receives instructions and data from memory.
- Address Register (AR): register of size n_{adr} bits, which supplies an address to memory.
- Program Counter (PC): register of size n_{adr} bits, which contains the address of the next instruction to be executed.
- Instruction Register (IR): register of size n_{op} bits, which stores the opcode portion of the instruction code fetched from memory.

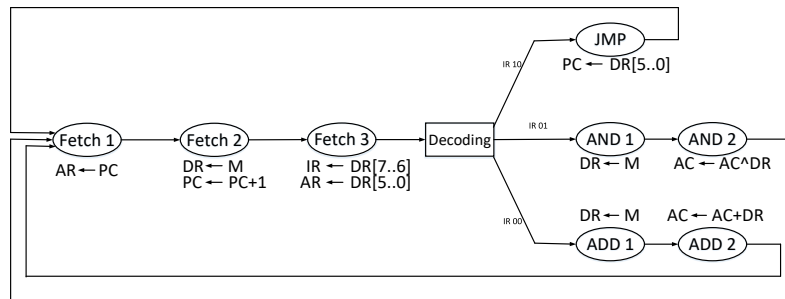


Figure 25: Instruction Cycle Example

Following is an explanation of the SEU propagation through the instruction cycle after causing a bit-flip in one of these 3 registers (PC, DR, AC, IR, and AR). To illustrate the processor’s modeling process, a small processor is considered, which can access 4 bytes of memory. Each instruction is a word consisting of a 2-bit operation code and a 6-bit address. This example assumes registers of the following lengths: DR: 8-bits; AC: 8-bits; AR: 6-bits; PC: 6-bits; IR: 2-bits. Three possible instruction codes are considered, shown in Table 8. The instruction code 11 represents an invalid operation. A possible progression of the processor’s instruction cycle is shown in Fig. 25.

6.3.1 Fetching Phase

This phase consists in obtaining an instruction from the memory and storing it in the appropriate registers. First, the contents of the program counter are stored in the address register (*fetch 1*). Next, the control unit reads the instruction from the memory. The control unit asserts a READ signal which causes the memory to output the requested data, which is stored in the DR, followed by incrementing the PC (*fetch 2*). Finally, the control unit copies the higher-order bits of the DR to the IR and the lower-order bits of the DR to the AR (*fetch 3*). The SEU propagation in the fetching phase is shown in Fig. 63(a). Starting from state $S0$, which represents an error-free fetching, the following possible causes to originate an error in this phase are considered:

- *SEUs affecting the PC* - This event may take place after the PC register is updated, following the *fetch 1* phase. An SEU-induced error in the PC register can alter the address of the next instruction, which may result in wrong or invalid operations. The transition from state $S0$ to state $S1$ (wrong or invalid PC) represents the effect of SEU causing a bit-flip in the PC register. The rate λ_{pc} indicates the probability of occurrence of an SEU-induced error in the PC register, which will result in a faulty state ($F1$) with probability 1, since all further operations will be performed on incorrect data.
- *SEUs affecting the memory* - This event may take place during the *fetch 2* phase, resulting in an alteration in the data accessed from the memory and stored in the DR register. The transition from state $S0$ to state $S2$ represents the effect of SEU causing a bit-flip in the memory. The rate λ_m indicates the probability of a bit-flip in the memory, which results in a faulty state ($F1$) with probability 1.
- *SEUs affecting the DR* - This event may take place after the *fetch 3* phase, when the contents of DR are accessed in order to populate the IR and the AR

Table 8: Operations

Instruction	Instruction Code	Operation
ADD	00XXXXXX	$AC \leftarrow AC + M[XXXXXX]$
AND	01XXXXXX	$AC \leftarrow AC \wedge M[XXXXXX]$
JMP	10XXXXXX	GOTO [XXXXXX]

registers. The transition from state $S0$ to state $S3$ represents the effect of SEU causing a bit-flip in the DR register. The rate λ_{dr} indicates the probability of a bit-flip in the DR register, which can directly cause an alteration in the operation to be performed, or an alteration on the data, resulting in a faulty state ($F1$) with probability 1. The occurrence of SEUs in the DR register may also lead to invalid values being stored in the IR and AR registers (i.e., an invalid operation in the case of the IR, or an invalid or out-of-bounds address in the case of the AR). These phenomena have a probability of detection, which is represented by the transition from state $F1$ to state $S0$, with rate λ_{inv} .

The absence of SEU-induced errors during the *fetch* phase is represented by the transition from state $S0$ to state $S4$, with probability λ_{s_2} .

6.3.2 Decoding Phase

After fetching the instruction from the memory, the control unit must determine which operation has to be performed. The value in the instruction register determines which execute routine is invoked. The state diagram in Fig. 25 represents this as a series of branches from the end of the fetch routine to the individual execute routines. The SEU propagation in the *decoding* phase is shown in Fig. 63(b). The following possibilities as causes of error in this phase are considered:

- *SEU propagating from fetch*: The error generated by an SEU during the *fetch* phase will propagate to the *decode* phase. This is represented by the transition from state $F1$ (fetch error) to state $F2$ (decode error), with probability λ_{prop} . This propagation may happen when the effects of an SEU result in the occurrence of a valid yet erroneous instruction in the affected register.
- *SEUs affecting the IR*: The transition from state $S4$ to state $S5$, with probability λ_{ir} , represents the occurrence of SEUs causing bit-flips in the IR register. This event may have two possible outcomes: 1) the bit-flip may alter the operation stored in the IR register to an invalid operation. In this case, the transition from state $S5$ to state $S0$ takes place, with probability λ_{ir_inv} . This signifies an operation reset, after the system identifies the invalid operation stored in the IR register; 2) the bit-flip may alter the operation stored in the IR register to another valid operation. In this case, the transition from state $S5$ to faulty state $F2$ takes place, with probability $1 - \lambda_{ir_inv}$.

- *SEUs affecting the AR*: The transition from state $S4$ to state $S6$, with probability λ_{ar} , represents the occurrence of SEUs causing bit-flips in the AR register. This event may also have two possible outcomes: 1) the bit-flip may alter the value of the AR register to an invalid or out-of-bounds memory address. In this case, the system is able to identify the error and the reset transition, from state $S6$ to state $S0$, takes place with probability λ_{ar_inv} ; 2) the bit-flip may alter the value of AR to another valid memory address. In this case, the transition from state $S6$ to state $F2$ takes place, with probability $1 - \lambda_{ar_inv}$.

The absence of SEU-induced errors during the *decode* phase is represented by the transition from state $S4$ to state $S7$, with probability λ_{s1} .

6.3.3 Execution Phase

The last step of the instruction cycle is the execution of the decoded instruction, performed by the data path. At this phase, the data and the operation have been already fetched and decoded and are ready to be processed by the ALU. For multi-operand execution such as the *ADD* and *AND* as shown in Fig. 25, first one of the operands is fetched from the memory and must be stored in the AR (*ADD1*, *AND1*). Then, the data path performs the logical operation between the content of the AR and the content of the accumulator (AC). The result is stored in the AC, overwriting the previous value (*ADD2*, *AND2*). Then, the execution phase terminates and the next fetching phase begins. Alternatively, the single-operand executions such as *JUMP* (*JMP* in Fig. 25) operation is much simpler. It is implemented by fetching the address to which the processor must jump and copying it into the program counter. The error propagation in the *execution* phase is shown in Fig. 63(c). The following possibilities as causes of error in this phase are considered:

- *SEU propagation from decode phase*: It is considered that an SEU that was not detected during the decoding phase will be processed during the execution phase. This is represented by the transition from state $F2$ to state $F3$ (execution error), with probability λ_{prop2} . This generally means that the wrong operation was performed, or that the operation was performed on the wrong data. This SEU may also be logically masked in the data path. This is represented by the transition from state $F2$ to state $S7$ with probability $1 - \lambda_{prop2}$.

- *SEUs affecting the AC*: The transition from state $S7$ to state $S8$, with probability λ_{ac} , represents the occurrence of SEU in the AC register. As with previous registers, this event may have two possible outcomes. 1) the bit-flip in the AC register results in the output of erroneous data, as shown in the transition from state $S8$ to state $F3$, with probability λ_{prop3} . 2) this SEU may be logically masked, which is represented by the transition from state $S8$ to state $S7$, with probability $1 - \lambda_{prop3}$

The rate of a bit-flip due to SEUs in a register depends on the size of the register and static cross-section of each bit. Therefore, the rates in Fig. 63 are different for each register. It is also important to note that having an SEU at different registers has a different impact on the system behavior. In other words, the fail states in Fig. 63 (F0, F1, F2) will have different repercussions in different failure scenarios. For example, the bit-flip in the PC will most probably lead to wrong operation and wrong data, while a bit-flip in the DR register can only affect either the operator or the operand of the instruction.

6.3.4 Self-Repair Routine

In this work, it is assumed that the processor under test is equipped with a self-repair mechanism. The behavior of the self-repair routine of this processor can be represented as the finite-state machine (FSM) shown in Fig. 26. From an error-free operation state $R0$, the processor moves to state $R1$ after the occurrence of an SEU. At that point, the processor performs a fault detection routine. If the error is detected, then it will be fixed with probability λ_{repair} . The processor moves to state $R0$ (i.e., resuming error-free operation). From state $R1$, if the error could not be fixed within a certain time limit, the processor moves to state $R2$. The system eventually fixes this type of errors at the output by invoking a reset routine, which returns the system to state $R0$.

6.4 Proposed Formal Modeling and Analysis in PRISM

The modeling is done by generating different probabilistic automata (PA) to describe the behavior of SEUs propagating through the different registers, as well as the logic

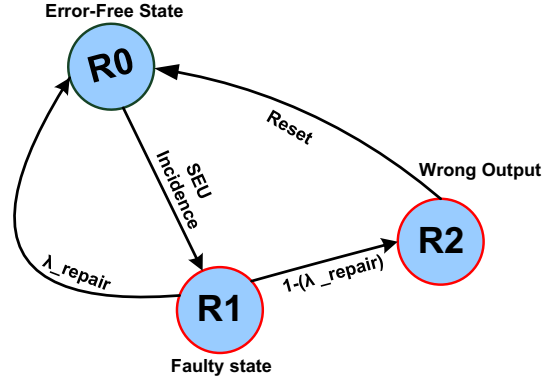


Figure 26: FSM of the Self-Repair Routine [9]

unit. The processor’s CTMC model is obtained by the parallel composition of all the PAs. The state transition probabilities are obtained by incorporating equations (1), (2), and (3) into the CTMC model. The propagation of SEUs through the processor is modeled as a CTMC, expressed in *Definition 1*.

Definition 1: A stochastic process $\{X(t) : t \geq 0\}$ with discrete state space S is called a continuous-time Markov chain if for all $t \geq 0$, $s \geq 0$, $i \in S$, $j \in S$, $P(X(s+t) = j | X(s) = i, \{X(u) : 0 \leq u < s\}) = P(X(s+t) = j | X(s) = i) = P_{ij}(t)$, where $P_{ij}(t)$ is the probability that the chain will be in state j , t time units from now, given it is in state i now.

The analysis of the processor’s CTMC is done in PRISM (Probabilistic Symbolic Model Checker) [91], a well established tool for formal modeling and verification of stochastic systems. A PRISM model is formed by basic constructs called modules, each designed to express a specific behavior, much like sub-components of a system. The state of each module is given by a set of finite ranged variables. The global state of the model is determined by the evaluation of the values of the module variables. Each module is composed of a set of commands, expressed in the format [$\langle act \rangle$] $\langle guard \rangle \rightarrow \langle rate \rangle : \langle action \rangle$, where:

- *act* is an action label used for synchronization of the different modules of the system;
- *guard* is a predicate over the operations performed in the system’s modules;

- *action* is a set of n updates that will translate into operations being executed in the modules.
- *rate* is the probability of occurrence of an action;

The commands are interpreted in a way that if the guard is satisfied, then the module is allowed to perform the corresponding transition with the associated rate. PRISM also allows the use of Markov reward structures. Reward structures are a way of extending a Markov chain by adding a reward rate to each state. This mechanism allows the modeling of variables that record the rewards accumulated with time. In this paper, the analysis of the system is performed by verifying a set of Continuous Stochastic Logic (CSL [19]) properties over the generated CTMC model. These properties can be used to obtain the probabilities of reaching a certain state in the model, as in (12), or to evaluate the expected value of a variable over a certain amount of time through reward structures, as in (5).

$$\mathbf{P}=?[\mathbf{F}(bit_n = seu \ \& \ out = seu)] \quad (4)$$

Which is interpreted as: “*what is the probability of eventually having a SEU in bit n that will eventually propagate to the output?*”.

$$\mathbf{R}\{“ctl_detected”\}=? [(C < T)] \quad (5)$$

Which is interpreted as: “*what are the accumulated rewards gained from reaching a state where an error has been detected in the control path during time T ?*”.

6.5 Experimental Analysis

In this section, the results of the analysis of a general accumulator-architecture processor at the system level are presented and discussed. The analysis has been performed on PRISM 4.3.1, running on a machine with an Intel Core I5-4200U CPU and 8 GB of RAM. In this work, the experiments were performed under the following assumptions:

- The considered analyses of SEUs occur in the PC, DR, AR, IR, and AC registers. Each register is 32-bit wide, and all bits from the registers have the same probability of being flipped. The processor has other special purpose registers,

such as the Stack Pointer (SP), Global Pointer (GP), Frame Pointer (FP), Return Register (RA), and Zero (always has the value 0). These special purpose registers are not considered in this analysis.

- At the system level, the exact flip rate of a bit due to SEUs is not obtainable because the hardware implementation of the system is not yet available. Therefore, it is assumed that the target system has a static cross-section per bit of $6.7 \times 10^{-15} \text{cm}^2/\text{bit}$.
- The rate at which the processor detects and recovers from errors depends on the adapted repair technique. Therefore, the repair rates used are an estimation.

The goal of the first experiment is to measure the impact of bit-flip injections on the different registers of the system. This is done to obtain the average expected time until failure, defined as Mean Time to Failure (MTTF), or the time from the injection of the SEU until the system reaches a fail state. For this purpose, SEUs are injected in each register separately in order to evaluate the MTTF. The results are shown in Fig. 27, where *Invalid Operation* indicates the MTTF of an error that will lead to an invalid operand. *Valid but Wrong Operation* is the MTTF of an error that generates a wrong but valid operator (i.e., change from addition to jump). Lastly, *Wrong Data* indicates the MTTF of an error that will lead to a wrong operand. It is important to note that higher values of MTTF contribute positively to the system's reliability, since this means that the system stays in a non-faulty state for a longer period of time before an error occurs.

It can be observed that the expected MTTF varies for different registers. For example, the MTTF of all the SEU injection scenarios in the DR is less than the MTTF of the corresponding injection scenarios in the PC. Moreover, it is noticeable that bit-flips affecting the operator of an instruction always result in lower MTTF compared to bit-flips that cause errors on the operands. This indicates that bit-flips propagating through the control path are more critical. This is mainly due to the fact that bit-flips that alter any of the bits dedicated to the identification of the operator will often result in invalid or wrong operators. If the operator is invalid, the system will detect the error immediately and it will reset the instruction, resulting in a low MTTF. If the operator is altered to a valid but wrong operation, it is assumed that the system will immediately enter a fail state, also resulting in low MTTF, but

the wrong operation is still carried out by the system. However, if an SEU alters one of the operands, it is not yet considered as a failure. This is because such SEU can be logically masked in the data path as explained in the execution phase of the instruction cycle, in Section 6.3.3. Therefore, such SEU will not be detected until the wrong output is generated.

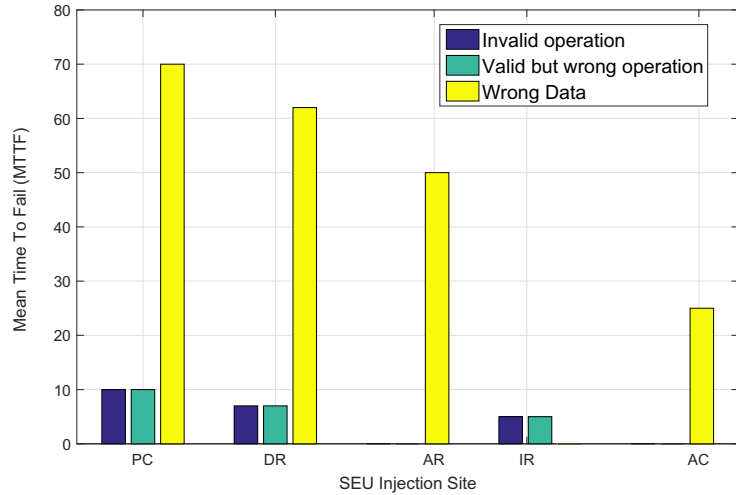


Figure 27: Mean Time to Failure

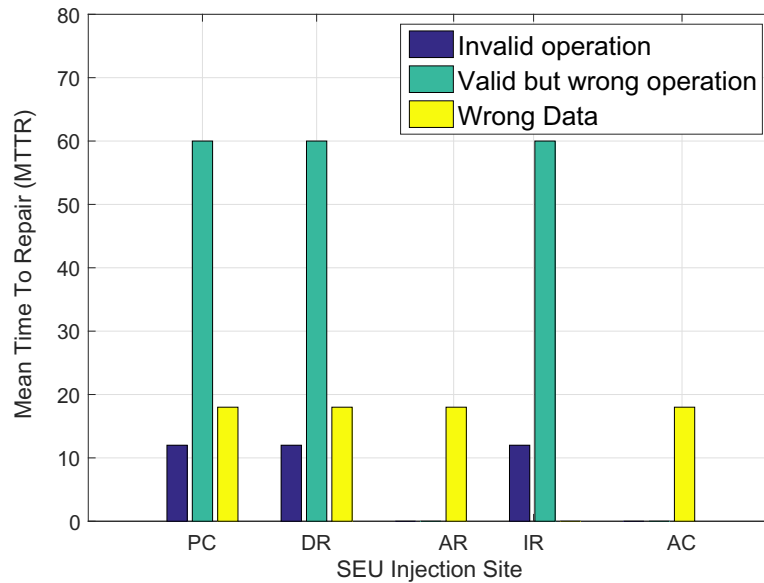


Figure 28: Mean Time to Recover

In the second experiment, the starting assumption is that the processor has failed (i.e., a bit-flip has been detected) and an analysis of the average time needed to resume the correct operation is performed. This is defined as Mean Time To Recover

(MTTR). From this definition, it can be concluded that lower values of MTTR contribute positively to the system’s reliability, since this means that the system spends less time in a non-functional state. From the results in Fig. 28, it is observed that the location where the bit-flip occurs has a significant impact on the MTTR. The MTTR value for an SEU inducing an *Invalid Operation* is low, since the operation reset is triggered immediately after the system detects the presence of the invalid operator. It can be observed that the MTTR value for an SEU inducing a *Wrong Data* error is also low. This is caused due to the fact that the wrong data error is not identified by the system during the instruction cycle. This error is identified after the result is generated. At that point, the system determines that the result is wrong and immediately performs a reset operation. It is interesting to note that the *Valid but Wrong Operation* error has the highest MTTR observed in the experiments. This can be explained by the fact that from the moment at which the wrong but valid operator is generated, the system is considered to be faulty. In other words, the operation has to be completed and the result has to be generated before the system is able to identify the error and perform a reset. Therefore, it can be concluded that SEUs inducing *Valid but Wrong Operation* errors are the most critical when considering MTTR.

Through the results obtained in the two previous experiments, it is possible to compute the limit of the availability function of the system as time tends to infinity, defined as the steady state availability (SSA). The SSA is given by $\frac{MTTF}{MTTF+MTTR}$ and it is automatically computed for each SEU injection scenario, as shown in Fig. 29. It is important to note that a higher SSA effectively means a safer system over time. A safer system is one with the higher MTTF (i.e., the system operates for a longer period) and lower MTTR (i.e., the system spends less time in a non-functional state). Thus, it can be concluded from the results that SEUs which induce the generation of valid but wrong operators are the most critical ones, since these are the SEUs that lead to the lowest SSA values. It can also be observed that SEUs that affect data bits are relatively the least harmful to the system, since a wrong operand can still produce a correct result due to the masking effects. This is evidenced by the fact that *Wrong Data* has the highest SSA across all different registers. The *Invalid Operation* errors can be considered to be of average criticality, since despite having a low MTTF, they also have a low MTTR.

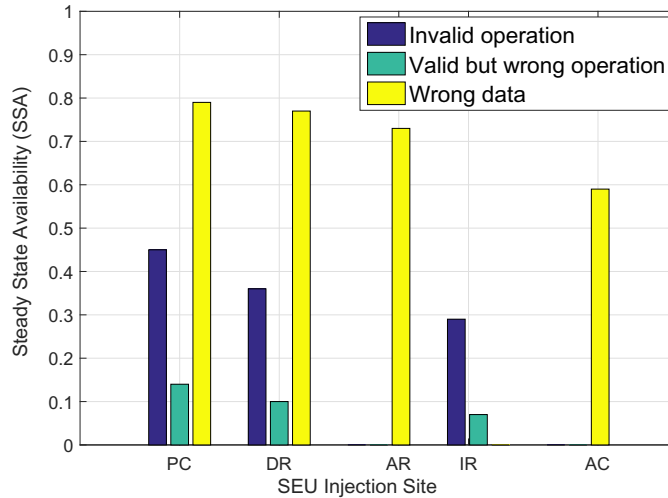


Figure 29: Steady State Availability

The premise of the fourth experiment is to take advantage of the proposed probabilistic system-level analysis to obtain the required self-repair rate of the system for different SEU-induced bit-flip rates. The parameters of this experiment are:

- 1) The failure rate of the system due to SEUs should be kept below a certain threshold. In this experiment, this threshold is considered to be equal to 1.5×10^{-16} .
- 2) Each register has an SEU-induced bit-flip rate that increases over time, between the values of 2.0×10^{-15} and 7.0×10^{-15} .

In this experiment, for each bit-flip rate due to SEUs, an investigation of the required self-repair rate in each of the analyzed registers in order to keep the system's reliability above threshold is performed. The results of this experiment can be observed in Fig. 30, where each line shows the relation between the SEU bit-flip rate in a register and the required self-repair probability. While these results may seem trivial at first (a higher bit-flip rate requires a higher self-repair probability), the experiment succeeds in showing one of the strong points in the proposed approach. For any given radiation scenario stipulated by a designer, the proposed model is capable of giving the optimal repair rate required for the system to remain within the predefined threshold of reliability.

In order to estimate the accuracy of the results provided by the proposed model, an additional experiment has been performed. This experiment compares data obtained from the work in [57] with data obtained from the proposed model, under the same conditions. The experiment is summarized in Table 9. For each test scenario (Sc.1, Sc.2, Sc.3), the rates for errors detected (P_D), errors not detected (P_{ND}), and errors

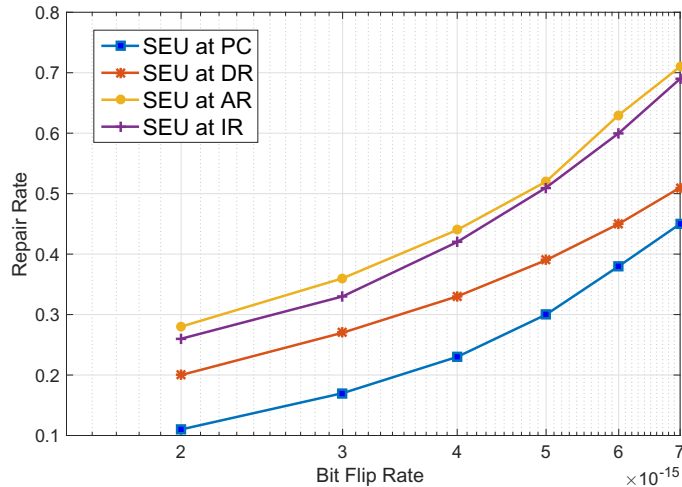


Figure 30: Vulnerability of Different Registers

detected at the output (N_{det}) are taken from [57] and used to build a custom test environment, as shown previously in Section 6.2, through Equations (1), (2), and (3). In this experiment, it is assumed that all SEU injections occur in the PC register. In [57], for each scenario, two values are given for the MTTF and for the MTTR. The first value is obtained through emulation testing, shown in Table 9 under *Emulation*. The second value, obtained through simulation at RTL-level, is shown in the table under *Simulation*. The results obtained by the model proposed in this paper are reported in the table under *Proposed Technique*. From the table, it can be seen that the results obtained when injecting SEUs in the PC register with the proposed system-level model are very consistent with those obtained with the RTL model introduced in [57]. For instance, in Sc.1 the MTTF and MTTR reported in [57] differ from our results by 1.48% and 1.61% respectively. Similarly, the differences obtained in Sc.2 are of 1.89% and 2.01% for the MTTF and the MTTR, respectively. Finally, the differences obtained in Sc.3 are of 2.21% and 2.46% for the MTTF and the MTTR, respectively. This is remarkable as, in our case, faults are injected at each bit of the PC register, whereas in [57], faults are injected as bitflips in memory. Thus the two experiments are not equivalent and differences should not be interpreted as modeling errors. Note also that SEUs injected in other registers with our method lead to other MTTR and MTTF values for which no comparable results are available in [57]. Furthermore, it is not clear that injecting faults in memory could reflect the MTTR and MTTF that we compute (see Fig. 28, 27, 29, and 30). This is not surprising as our method has a much better controllability and observability of the internals of our

model that methods such as [57].

Table 9: MTTR and MTTF in Different Techniques

	MTTF (seconds)			MTTR (seconds)		
	[57]		Proposed Technique	[57]		Proposed Technique
	Emulation	Simulation		Emulation	Simulation	
Sc. 1	77.8	77.3	76.7	105.8	106.8	104.1
Sc. 2	64.5	64.7	63.3	82.9	83.7	81.3
Sc. 3	64.5	64.4	63.0	62.8	63.9	61.2

Table 10: AVR ATmega103 Analysis Comparison

	Contribution to Microcontroller Failure						AnalysisTime
	Register File	ALU	PC	IR	Internal SRAM	Data Bus	
Proposed Technique	22.2%	24.5%	20.3%	18.7%	14.3%	-	3.2 Seconds
[118]	23.8%	26%	19.85%	17.16%	13.41%	10.21%	4 hours

The final experiment conducted seeks to demonstrate the efficiency of the proposed approach when compared to existing fault injection through simulation techniques. Fault injection through simulation requires huge computational power and time and it is a common occurrence for such techniques to take several days to conclude the analysis, depending on the parameters and the complexity of the analyses. Our work tries to provide an answer to this problem, by introducing a very fast and resource-friendly approach to fault-injection experiments. In addition to the proposed system level probabilistic abstraction, this improvement in processing time is possible thanks to two main factors: 1) The number of fault injection scenarios required to estimate the system vulnerability to SEUs is reduced: in large systems, many faults injected at different bits may explore the same propagation paths. Traditionally, the faults are injected into the system and the results are monitored. In other words, the design is treated as a black box [17]. However, the proposed methodology grants us a higher level of control over the fault injections. Thus, all redundant fault propagation paths can be reduced to optimize the number of cycles required to analyze all faults in the system. 2) The time required to analyze each injection scenario is reduced as the proposed probabilistic model is heavily optimized for state-space reduction. This is achieved by analyzing the fault propagation paths originating from each injection scenario and deactivate nodes that have no impact on the analysis. For example, during the analysis of faults injected at the Decode Cycle, the states of the components that are part of the Fetch Cycle have no impact on the outcome. This results in a

technique that is several orders of magnitude faster than fault injection approaches based on simulations. The work in [118] presents a technique for reducing CPU time to perform simulation-based fault-injection experiments in complex System on Chips (SoCs). The goal of this final experiment is to draw a direct comparison with the results and analysis time required by the technique introduced in [118]. This was done by applying the concepts of the proposed technique to model and analyze the core of the AVR microcontroller ATmega103 [3], and comparing the data obtained from the work in [118] with data obtained from the proposed model, under the same conditions. In this experiment, we have injected faults at different points during the ATmega103's instruction cycle in order to reproduce the injection scenarios used in [118], and to estimate the contribution of different subcomponents to produce a failure in the microcontroller. The results in Table 16 show that the proposed technique provides consistent results while being orders of magnitude faster.

6.6 Conclusion and Future Work

This paper presents a novel approach for dependability estimation of fault tolerant systems exposed to SEUs, based on probabilistic model checking. This approach seeks to overcome the limitations of emulation and simulation-based techniques, by providing a fast and exhaustive analysis of the effects of soft-errors in the system. The experimental evaluation conducted is able to accurately quantify the impact of SEUs on the different registers considered and through the different paths of error-propagation in each SEU injection scenario. The obtained results are used to provide an estimate of a processor mean time to failure and mean time to recover values. Then, these values are used to compute an estimation of the steady-state availability of the system, or the limit of the system's availability over time. Finally, a dynamic test scenario is presented, where the technique is able to estimate the required self repair capabilities of the system in order to maintain its failure rate under a predetermined threshold. Future work may analyze more complex processors, as well as different architectures and different types of transient errors.

Chapter 7

Article IV: Reliability Analysis of the SPARC V8 Architecture via Fault Trees and UPPAL-SMC

Authors: Marwan Ammar, Ghaith Bany Hamad, Otmane Ait Mohamed, Yvon Savaria

Abstract: This paper proposes a system-level dynamic fault tree approach to model, analyze, and estimate the vulnerability to soft-faults of the 7-stage pipelined SPARC V8 integer unit. A preliminary analysis of the architecture is used to derive a dynamic fault tree diagram which is modeled as a priced-timed automaton. The assessment of the architecture's vulnerability to radiation effects is obtained through fault tree analysis, showing consistent results with radiation and simulation tests.

7.1 Introduction

The cost and the complexity involved in the development of safety-critical systems are a prime motivator to the use of reliability assessment techniques as early in the design cycle as possible. Existing techniques often lack the capacity to perform a comprehensive and exhaustive analysis on complex architectures exposed to Single Event Upsets (SEUs), leading to the necessity of conducting expensive ground tests that are not able to fully characterize the system's vulnerabilities. An SEU is characterized by an

unforeseen change of state in one or more elements within the system memory. That change of state is known as a soft-fault, and it can often be detected and corrected by safety measures designed in the system. However, failing to detect the presence of soft-faults may have catastrophic consequences, especially in environments where safety is paramount.

In recent years, as the demand for fast, flexible and reliable techniques have increased, many alternative methods for soft-fault estimation have emerged. Among the most popular techniques in the literature is the fault tree analysis (FTA). The FTA method is widely used for risk assessment, mainly in the area of avionics, nuclear and chemical industries [138]. Dynamic Fault Trees (DFTs) [49, 138] are an extension of static fault trees that cater to complex functional dependencies between system components, such as priority, order of occurrence, triggering, and spare component management. Recent literature such as the work in [120] propose extensions to the conventional boolean FTA in order to take sequence dependencies into account for qualitative and probabilistic analyses without state-space transformations. This allows for modelling of event sequences at all levels within a fault tree. The analysis of uncertainty over time is the focus in [10], in which a FTA technique has been introduced to accurately model the effects of SEUs in non-deterministic environments, with the use of probabilistic model-checking and Markov Decision Processes (MDPs).

This paper introduces a new DFT approach to compute an accurate estimation of a system's vulnerability to soft-faults, using the 32-bit SPARC V8 integer pipeline as a case-study. The fault propagation paths in the targeted architecture are obtained from the SPARC V8 architecture manual [74] by applying the technique introduced in [11]. Subsequently, the fault propagation paths have been combined in order to obtain a representative FT of the SPARC V8 integer pipeline, through the use of the *Behaviour-Based Method* [112]. Next, a new modeling of FT gates is proposed, using the Priced-Timed Automata theory. Finally, a fully automatic FTA is performed with the use of UPPAAL-SMC model checking. The analysis performed consists in the estimation of the probability of each type of Trap Exception (TE) to occur in the targeted architecture, as well as the impact of individual registers to the overall reliability of the processor, and the probability of failures over time. The obtained results are compared with radiation and simulation tests, showing remarkable consistency with radiation and simulation tests.

7.2 Existing Fault Analysis of SPARC V8

Extensive literature exists on the verification of the SPARC V8 architecture, which is mainly performed through simulation and radiation testing. Radiation testing, such as the work in [127] are very important in characterizing this architecture’s vulnerability when exposed to total ionizing dose. In [29] the authors use software handlers that enable the classification of the types of crashes, and the measured crash cross-sections are compared with those predicted by fault injection simulation. This demonstrates that the data extracted from radiation tests may be used to conduct predictability experiments at a higher-level. However, these techniques are extremely costly and rather limited in their coverage, since it is virtually impossible to test or simulate for all possible scenarios. On the other hand, regular system-level approaches (such as FTs) are often bound to simplistic analyses due to a lack the expressiveness resultant from the high-levels of abstraction employed. To address these shortcomings, we propose a new time-enhanced modeling approach to FT gates. The proposed modeling enhances each of the gates with multiple clocks that may keep track of the duration of each fault individually, as well as global clocks that enable the analysis of the impacts of faults over time.

7.3 Modeling the SPARC V8 Pipeline as DFT

In order to conduct the proposed analysis, we must first generate a fault tree of the SPARC V8 integer pipeline. We have adopted an analytical approach for the generation of fault trees for complex systems, known as the *Behaviour-Based Method* [112] for the FT generation. This approach considers faults as behaviours, and fault-tree gates as operations on those behaviours. The behavioural patterns of the system under the effects of SEUs have been obtained by applying the method previously introduced in [11]. By applying these techniques to the SPARC V8 7-stage pipeline, and based on the structural information available in the SPARC V8 architecture manual [45, 74], the fault tree of the integer pipeline is constructed, as shown in Fig. 31. The fault tree is divided into 7 levels, each representing a stage of the pipeline. A more detailed explanation of how the FT was obtained, as well as the reasoning for the FT gates used is given in the following subsections.

7.3.1 General Considerations and Assumptions

Before discussing how the FT mapping has been done, it is important to present some of the general considerations and assumptions used in this work. Firstly, it is assumed that the outputs of the fault tree gates are also susceptible to soft-faults. For example, the probability of failure (λ) of “inst” is given by $(\lambda_{iCache} OR \lambda_{PC} OR \lambda_{inst})$. Secondly, we assume that the events in the fault tree (circles and rectangles) do not represent components but rather the occurrence of a soft-fault on the component [112]. For example, the element *Reg. File* means that a soft-fault has occurred in the component register file. Therefore, each event in the fault tree has a probability of failing due to soft-faults. Such probability is estimated through the equation: $\sigma_{device} = \sigma_{ff} \times N_{ff} \times (1 - \alpha)$, where $\sigma_{component}$ is the cross section of the device, σ_{ff} is the intrinsic cross section of the flip-flop mapped on the design, N_{ff} is the number of flip-flops, and α is the masking of the device [29]. Unlike conventional logic gates, the inputs and outputs of FT gates are probabilities related to the set operations of Boolean logic. For example, the *AND* gate represents the assumption of the combination of independent events (i.e., the intersection of the input event sets). On the other hand, the *OR* gate represents the assumption that the inputs are mutually exclusive events (i.e., the union of the input event sets) [138]. An important consideration is that some FT behavior has been slightly altered. For example, the register *inst* connects directly to three other registers (i.e., *imm*, *rfa*, or *rd*) through different transitions. Based on which bit of register *inst* is affected by the SEU, a different fault propagation path may take place. This decision is taken probabilistically.

7.3.2 System Level Fault Abstraction

This subsection details how each of the SPARC V8 pipeline stages have been mapped into a FT. This mapping has been based on extensive research on fault injection and propagation experiments in the literature, as well as in technical reports and in the SPARC V8 architecture manual.

Instruction Fetch (FE): In this stage, the PC register is read and the instruction is fetched from the instruction cache (*iCache*). Therefore, in this stage, a soft-fault may originate from the PC or from the *iCache*. This relationship is represented in the proposed FT as gate *G1*, which is an *OR* gate connected to the basic events *iCache*

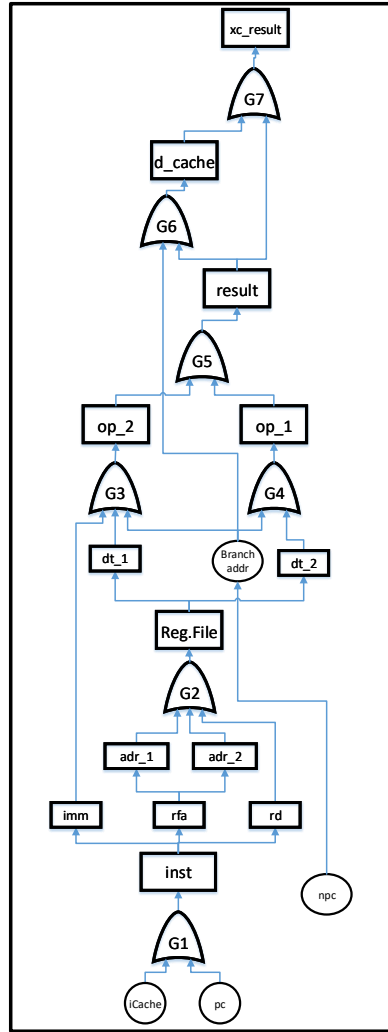


Figure 31: DFT of the 7-stage integer pipeline of the SPARC V8

and *PC*. Additionally, Soft-faults in the *nPC* components may be propagated to the *EX* stage of the pipeline.

Decode (DE): In this stage, the instruction (*inst*) is decoded. A soft-fault that occurs in the *inst* register can non-deterministically propagate the error to one of three registers: *imm*, *rfa*, or *rd*. Each of these registers will, in turn, activate a different fault propagation path. Register *imm* may propagate to the *Execute* stage. Register *rd* may propagate to the *RA* stage and register *rfa* can be non-deterministically propagated to *adr_1* or to *adr_2*.

Register Access (RA): During the register access stage, operands are read from the register file or from internal data bypasses. In our model, a soft-fault at this stage

(either direct or propagated) may firstly affect the registers adr_1 or adr_2 . These registers, along with the rd register from the DE stage, are inputs to the OR gate G2. A soft-fault in the inputs of gate G2 may propagate to the *Reg. File* component. It is important to note that the *Reg. File* component may propagate the soft-fault to either dt_1 or dt_2 . According to [74], a soft-fault in adr_1 may only propagate to dt_1 . Similarly, a soft-fault in adr_2 may only propagate to dt_2 . Therefore, the *Reg. File* component and its children are considered special cases, deviating from the previously mentioned general assumption since the choice of propagation path is always deterministic.

Execute (EX): During this pipeline stage, the logical and shift operations are performed. For memory operations (such as *JMPL*), the address is generated. In this stage, soft-faults originated from the imm , dt_2 , or $branch\ addr.$ may propagate to component op_1 through the OR gate G3. Similarly, soft-faults originated from the dt_1 or $branch\ addr.$ may propagate to component op_2 through the OR gate G4. Furthermore, soft-faults in op_1 , op_2 , or pc may propagate to the *result* register.

Memory Access(MA): Data cache is read or written in the memory at this stage. In the proposed FT, an error in this stage may arise either from a soft-fault in the *result* or in the $branch\ addr.$ registers. This is modeled with OR gate G6, which may propagate the soft-fault from either of those registers.

Exception (XC): In this pipeline stage, traps and interrupts are resolved. In the proposed FT, OR gate G7 may propagate soft-faults coming from the d_cache or from the *result*. At this level of the FT, our model computes the probabilities of each of the different traps, based on the probability of soft-faults in the instructions performed and the paths taken.

Write-Back (WB): The result of any ALU, logical, shift, or cache operations are written back to the register file. The value of the xc_result register is generated during this pipeline stage.

7.3.3 PTA Model Composition

The modeling formalism of UPPAAL-SMC is based on a stochastic interpretation and extension of the Timed Automata (TA) formalism used in the classical model checking version of UPPAAL. For individual TA components, the stochastic interpretation replaces the non-deterministic choices between multiple enabled transitions

by probabilistic choices. Similarly, the nondeterministic choices of time delays are refined by probability distributions, which at the component level are given either uniform distributions in cases with time-bounded delays or exponential distributions (with user-defined rates) in cases of unbounded delays. These structures are defined as Priced-Timed Automata (PTA).

In order to accurately model the fault dependencies in the proposed DFT, our approach focuses on the formalization and modeling of the probabilistic behavior of FT gates and events over time. To achieve this, we define each FT gate as a PTA model, where a state $(l, \nu) \in L \times \mathbb{R}_{\geq 0}^X$ such that $\nu \models \text{inv}(l)$. In any state (l, ν) , there is a nondeterministic choice of either making a discrete transition or letting time pass. A discrete transition can be made according to any $(l, g, p) \in P$, with current state l being enabled and zone g is satisfied by the current clock valuation ν . The probability of moving to location l' and resetting all clocks in X to 0 is given by $p(X, l')$. The option of letting time pass is available only if the invariant condition $\text{inv}(l)$ is satisfied while time elapses. The complete model of the desired FT can be obtained by synchronizing the inputs and outputs of the required FT gates, therefore composing the full FT diagram. As an example, Fig. 32 shows the PTA of a possible configuration of the *OR* gate with two inputs. In the case of the x input, it may fail at any time with probability px . At which point, the variable x becomes 1. The fault in x then propagates to the output of the gate, setting out to 1. The same logic also applies to the y input.

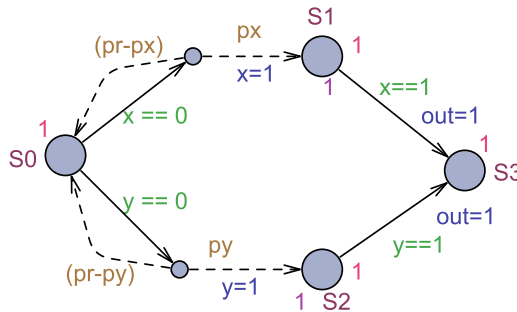


Figure 32: Sample of an OR Gate PTA

7.4 Stochastic Soft-fault Analysis with UPPAAL

In this section, we present and discuss the results of our analysis of the fault tree of the SPARC V8 pipeline (Fig. 31). For this analysis, the proposed FT has been

modeled in Uppaal-SMC, where a PTA model of each of the FT gates has been generated. The model of the FT is then obtained through the parallel composition of all the gate models. The proposed PTA models in Uppaal rely heavily on two aspects: 1) As previously mentioned, the probabilities used for the failure rates in the model are derived from the cross-section values reported in [29]; 2) The proposed DFT models are constructed specifically for the verification of a microchip. This means that the PTAs of each gate of the FT have certain configurations that accommodate the particularities of such system. For example, *exit rates* in the models are set to 1. This forces the tool to evaluate each state at every unit of time (clock-cycle). Furthermore, each register has an internal clock that tracks the amount of time that the soft-fault is active, as well as a global clock that estimates the average propagation times in the FT. This feature is extremely valuable for the modeling of soft-faults, since it allows the model-checker to verify the model with different time parameters in each verification instance. For example, the expected propagation time of a fault greatly impacts its probability to cause a failure in the system.

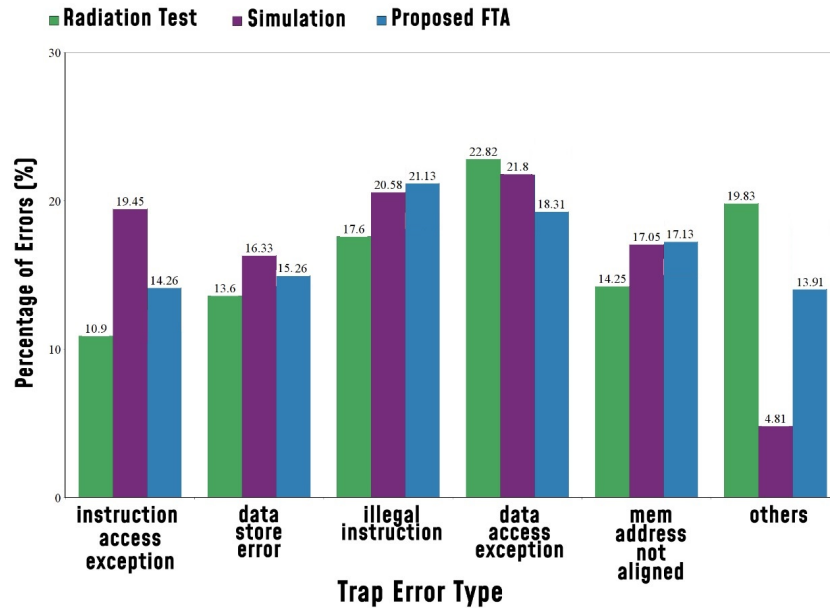


Figure 33: Probability of Trap Exceptions in different approaches. Simulation and radiation test results are reproduced from [29]

This experiment has the objective to determine the probability of an SEU to rise a *trap exception* in the pipeline of the processor. Moreover, the proposed analysis can identify the type of trap exception that has occurred. With the FT proposed in Fig. 31 and following the behavior detailed in the SPARC V8 manual [74],

the proposed model is able to estimate the probability of occurrence of the following trap exceptions: *instruction_access_exception*, *data_store_error*, *illegal_instruction*, *data_access_exception*, and *mem_address_not_aligned*. All other types of trap exception are classified under *Others*. The probabilities used in the proposed analysis are derived from cross-section metrics published in [29]. The analysis has been evaluated over 3800 iterations, with a confidence level of 95%. The results show that approximately 41% of all injected faults have been captured as trap exceptions. Fig. 48 shows a break down of each of the identified errors. For validation, the obtained results (shown under *Proposed FTA*) are compared with the results reported in [29] (shown under *Radiation Test* and *Simulation*). It can be observed that the results estimated by the proposed analysis show remarkable similarity with the values obtained by the of the *Radiation Test* and *Simulation* techniques. This shows that the proposed analysis can be a viable option for early analysis of microprocessor designs, as it is an inexpensive, fast, and highly customizable alternative to other adopted methods.

Table 11: SPARC V8 Probability of Failure Over Time

	Prob. of TE (million hours)	Prob. of Error (million hours)	Prob. of Undetected Errors (million hours)
SPARC V8 Pipeline	0.133	0.268	0.097

The proposed FTA can be customized to evaluate other metrics, such as the estimated time before a failure, the failure rate over time, and the impact of soft-errors on different components. As an example, Table 16 shows the probability of trap exceptions over time, the probability of detected errors over time, and the probability of undetected errors over time. Although relatively small, the probability of undetected errors in the system may represent a serious issue in certain conditions, where the system is expected to operate without maintenance. Our analysis shows that the biggest contributors to the occurrence of undetected errors are the nPC register (27.2 % of cases), the rfa register (22.3 % of cases), and the d_cache (19.3 % of cases). The access to this kind of information early in the development cycle means that possible points of vulnerability are easier to detect and quicker to fix, resulting in increased productivity at lower costs.

7.5 Conclusion and Future Work

Vulnerability to soft-errors is a major concern for micro-electronic systems exposed to radiation. This paper proposes a probabilistic verification approach for vulnerability estimation of the SPARC V8 architecture, when it is exposed to soft-errors. This approach seeks to overcome the limitations of emulation and simulation-based techniques, by performing fault tree analysis through stochastic model checking, which provides an accurate and exhaustive estimation of the effects of soft-errors in the system. The modeling experiments that were conducted produced results that are very consistent with previously reported radiation ground testing. The proposed approach is also able to accurately estimate metrics such as the impact of different bit flips on the system's dependability measurement, and availability over time. Future work will seek to expand the analysis domain by integrating multiple layers of abstraction in order to produce even more accurate results.

Chapter 8

Article V: Towards an Accurate Probabilistic Modeling and Statistical Analysis of Temporal Faults via Temporal Dynamic Fault-Trees (TDFTs)

Authors: Marwan Ammar, Ghaith Bany Hamad, Otmane Ait Mohamed, Yvon Savaria

Abstract: Fault Tree (FT) is a standardized notation for representing relationships between a system's reliability and the faults and/or the events associated with it. However, the existing FT fault models are only capable of portraying permanent events in the system. This is a major hindrance since these models fail to reflect accurately the other classes of faults, such as soft-faults, which are often temporary events that usually disappear after the source of the interference is no longer present. This paper proposes a new fault tree modeling paradigm, to capture the impact of temporal events in systems, called *Temporal Dynamic Fault Trees (TDFTs)*. TDFTs are utilized to model the characteristics and dependencies between different temporal events, soft-faults, and permanent faults. These features are integrated to the proposed probabilistic models of the temporal gates, which are modeled as *Priced-Timed*

Automata (PTA). This paper also proposes a new FT analysis methodology, based on *Statistical Model Checking* (SMC), designed to circumvent the state-explosion problem that is inherent to other model-checking approaches. The proposed analysis is able to evaluate the impact of temporal faults in systems, as well as to estimate the reliability and availability of the system over extended periods of time. The experiments reported in this paper demonstrate the versatility and scalability of the proposed approach. For instance, the results display the impact that temporal events may have in a digital system. Our observations indicate that while regular soft-fault analyses tend to underestimate metrics such as system reliability, TDFT analysis shows remarkable consistency with radiation testing, with differences of under 2%, in the conducted analysis.

Index Terms: Fault Tree, Temporal Events, Radiation effects, Single-Event Effects, Statistical Model-Checking, Formal verification, System-Level Analysis, Reliability, Availability

8.1 Introduction

Fault tree analysis (FTA) is a prominent fault diagnosis technique which has gained widespread acceptance for quantitative safety analysis. A fault tree consists of a diagram which represents the failure of the top level event (TLE) according to the failure of basic events based on the relationships between them. The objective of FTA is to provide insightful information to designers regarding the reliability of their systems by identifying the ways in which the system is most likely to fail and thus showing the most efficient ways to make the system safer. In the literature, FTs are often classified as either static or dynamic, based on the dependency relationships between their respective components and events. Examples of such dependencies can be event priority, sequence, spare behavior, etc. With the introduction of dynamic gates [30, 137, 143], the relationship between the events and components of a fault tree have changed into a dynamic one, in which the outcome becomes dependent on the order and the number of occurrences of basic events. Further development into dynamic gates has led to the development of fault trees with temporal requirements

[142, 144, 146]. These fault trees extend the dynamic gates to include tighter sequence requirements for the events. However, the inclusion of temporal constraints in these approaches greatly limits the analysis process. Traditionally, FTA is performed through simulation techniques [119], with tools based on techniques such as *Monte-Carlo simulation*, *time-sequential simulation*, and *discrete event simulation* [52, 113]. Analysis of large systems using these techniques can often be overwhelming for the tool since simulation-based analysis relies on input space sampling. This means that unless all possible points are sampled, there exists a possibility that an error is not detected by the analysis.

In recent years, many formal-based approaches for the analysis of dynamic fault trees have emerged [13, 47, 119, 139]. These approaches (detailed in Section 8.2) solve most of the limitations of simulation techniques, achieving fast and efficient FTA. However, current FTA approaches are not suitable for safety-critical analysis, since current FT modeling techniques cannot capture sequences of actions (such as how many times has event 1 occurred before event 2) and state history. Moreover, the binary representation (working or failed state) of FTs is not adequate for systems with complex state-spaces. For example, conventional fault tree events start as inactive and may become active according to a probability rate. Once an event becomes active, it will remain active throughout the rest of the analysis. This behavior makes it impossible to represent the occurrence of transient faults (i.e., faults that may appear or disappear from the system due to the effects of external sources of interference, such as radiation and heat) with existing fault-tree gates. This is due to the absence of suitable fine-grain management of time in conventional fault-tree structures [104].

This paper proposes the modeling of a new type of dynamic fault trees with strict behavioral and temporal requirements, hence referred to as *Temporal Dynamic Fault Trees (TDFTs)*. TDFTs are primarily targeted towards the analysis of temporal events, such as radiation effects and heat. However, the TDFT gates are flexible enough to be configured for the analysis of most FT systems. The work in this paper is distinct from the literature in the following ways: 1) The proposed TDFTs are fault trees that capture temporal events, state history and sequences of actions. We present and explain the models of each TDFT gate, formulated as Priced-Timed Automata (PTA). Thereafter, the complete model of the fault tree is obtained through the parallel composition and synchronization of the PTAs of all the required gates.

2) Each of the gates in the fault tree is extended with a unique clock which, along with a global system clock, enables precise time tracking and management. This allows the proposed model to accurately represent temporal faults, which are only active for a certain amount of time [103]. It also enables the verification of other temporal properties, such as the time required for a specific event to manifest itself in the system. 3) An analysis methodology is introduced, utilizing FT modularity and sequential hypothesis testing in order to decrease time and resource requirements while maintaining a high confidence level for the results. This is combined with *Statistical Model-Checking* (SMC), which provides statistical evidence for the satisfaction or violation of the specification.

The aforementioned distinctions are demonstrated through a comparative study, derived from verifying the estimated system reliability obtained with conventional FTs versus the new TDFT model. The proposed TDFT analysis is experimentally evaluated on different scenarios in order to assess its impact on the different types of fault tree gates and to demonstrate its versatility. Finally, the proposed methodology is used to analyze the 7-stage integer pipeline of a Leon-3 microprocessor. The obtained results are compared with radiation and simulation tests from the literature [29]. Our results demonstrate that regular FTA methods are inadequate for the analysis of systems that are exposed to temporal faults. The rest of this paper is structured as follows: In Section 8.2, some of the most relevant related works are briefly discussed. In Section 8.3, we explain a few preliminary concepts that are key to the development of this work. Section 8.4 introduces the proposed modeling of the TDFT gates, and the SMC-based analysis methodology. In Section 8.5, several experiments are presented. These experiments have the goal of demonstrating the main differences and advantages introduced by the proposed methodology. Finally, in Section 8.6, we draw some conclusions and discuss future works.

8.2 Related Works

FTA is an extremely important field of research. Being the focus of many groups during the past decade, FTA has evolved into one of the de facto techniques for early analysis of critical systems. Walker and Papadopoulos [144] first suggested extending static FTs with Priority-AND, Priority-OR, and Simultaneous-AND gates.

These gates enable a fault tree to enforce temporal dependencies between events. The works in [13, 139] present very robust formal approaches to DFT analysis, based on *Input/Output-Interactive Markov Chains* (I/O-IMCs) and stochastic model checking. The work in those papers apply modularization approaches and compositional aggregation techniques, along with heavy reduction algorithms, to formally analyze DFTs with model checking. These techniques are built around the *Galileo* formalism [128], which greatly limits the expressiveness of the analyzed models. For example, neither approach is able to handle self-repairable systems or any dependencies between gates or events that have not been pre-programmed in the tools.

In recent years, several techniques have been proposed to extend DFTs in ways to offer more flexible temporal dependencies between its faults and events. In [120], the author proposes an extension of the conventional boolean FTA in order to take sequence dependencies into account for qualitative and probabilistic analyses without state-space transformations. This allows modeling of sequences of events in all levels of the fault tree. The analysis of uncertainty over time is the focus in [81]. This work uses the Pandora tool and fuzzy logic in order to predict and capture different sequences of dependent dynamic events over time. The authors use their method to combine probabilistic data and fuzzy set theory with Pandora TFTs to enable dynamic analysis of complex systems with limited or absent exact quantitative data. Peng et al. [109] use a timed FT extension method applied to a railway maintenance system in order to identify which faults are likely to occur first and, therefore, must be eliminated more urgently. This method can estimate the time required for railway maintenance and thereby improve maintenance efficiency, and reduce risks.

All these techniques have in common the attempt to introduce new dependencies between the different components and events of dynamic fault trees. However, they lack the expressiveness required for a robust analysis of the impact of failures over time in a dynamic environment. The work presented in this paper further augments fault trees, by combining and extending dynamic, repairable and temporal fault trees. The proposed fault tree models are able to capture the randomness of fault testing, where a significant event (i.e., radiation) may occur non-deterministically. Furthermore, this event may be permanent, intermittent, or it may be active for a variable unknown amount of time.

8.3 Preliminaries

8.3.1 The UPPAAL Formalism

UPPAAL is a toolbox for verification of real-time systems, represented by a network of timed automata, extended with integer variables, structured data types, and channel synchronization. For the efficient analysis of probabilistic performance properties, UPPAAL-SMC proposes to work with *Statistical Model Checking* (SMC). SMC works by monitoring some simulations of the system, and then use statistical results (including *sequential hypothesis testing* or *Monte Carlo simulations*) to decide whether the system satisfies some property with a sufficient degree of confidence. The modeling formalism of UPPAAL-SMC is based on a stochastic interpretation and an extension of the *Timed Automata* (TA) formalism used in the classical model checking version of UPPAAL. For individual TA components, the stochastic interpretation replaces the non-deterministic choices between multiple transitions enabled by probabilistic choices (that may or may not be user-defined). Similarly, the nondeterministic choices of time delays are refined by probability distributions, which at the component level are given either uniform distributions in cases with time-bounded delays or exponential distributions in cases of unbounded delays [46].

An illustrative example of the UPPAAL formalism is given by the PTA in Fig. 34. In this example and through the rest of this paper, the weight annotations on locations and edges are ignored and defaulted to "1". For Fig. 34, the delay distribution determined by the upper and lower paths to the END state is given by sums of uniform distributions, where $X \geq 2$ (green label) is the guard of the transition (i.e., minimum time), and $X \leq 4$ (purple label) is the invariant distribution (i.e., maximum time delay) of the transition. The stochastic choice that determines which path will be taken is represented by a forked transition, where each path is weighted accordingly. In the example, the weights of each path are either $\frac{1}{6}$ or $\frac{5}{6}$. Finally, an update may be performed during each transition (blue labels). Therefore, the END location in the example is reachable within the interval $X = [4, 12]$.

8.3.2 Fault Tree Analysis

The fault tree analysis (FTA) method is a widely used method for risk assessment, mainly in the area of avionics, nuclear and chemical industries [137]. FTA follows

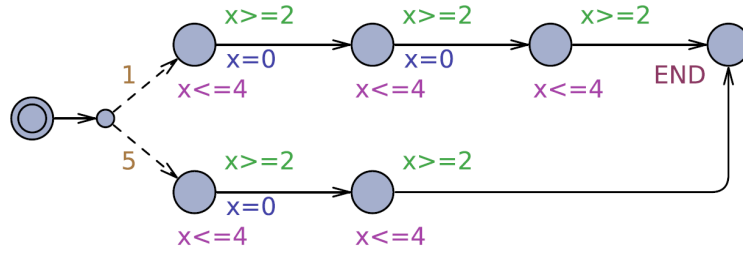


Figure 34: Illustrative Example of the UPPAAL Formalism. Reproduced from [46].

a deductive approach, which means that it starts from an undesirable general event in order to find what circumstances may lead to that event. In the context of FTA, the general event is known as the *top event*, from which the fault tree branches out vertically. The top event is defined as the failing point of a system in operating conditions, whether those conditions are considered normal or abnormal. Bottom events are occurrences that may lead to a component failure. As such, fault tree models are characterized as graphical representations of system failures, in terms of the system’s components. Standard fault tree models are defined as combinatorial models, composed by static gates (mainly *AND* and *OR* gates) and basic events. Combinatorial models can only handle combinations of events and not the order of occurrence of such events, thus are not able to represent complex systems adequately. Dynamic fault trees (DFTs) extend standard fault trees to allow the representation of more complex relationships between basic events, such as functional dependencies, priority, and order of occurrence.

8.4 Proposed TDFT Modeling and Analysis Methodology

A constant among existing FTA techniques is that the failure of a basic event cannot be reverted (i.e., when a basic event changes from the normal state to the fail state, it will remain in the fail state forever) [39]. However, it has been noted in the literature that basic events (i.e., events that may lead to the failure of a component) are not always permanent. This fact may heavily impact the results of fault analyses, especially in systems exposed to nondeterministic environmental interferences, such as radiation, heat, and cosmic rays [5, 80, 84]. In a previous work related to malfunction in pacemakers exposed to ionizing radiation [21], we have demonstrated that

these radiation-induced malfunctions vary from a simple temporary abnormality to complete system malfunction. The analyses in [21] have indicated that a simple temporary bit-flip (which, according to technical reports, is a common occurrence) may cause the pacemaker to behave erroneously and even endanger the life of a patient, in some cases.

The approach proposed in this paper overcomes this FTA limitation by introducing the sensitivity to *Temporal Basic Events* (TBE) to the fault tree gates (i.e, events that may appear for a limited amount of time and then disappear if certain conditions are met). This enables a more accurate representation of the environmental hazards that the system may be exposed to, as well as for the tracking of faults which may only manifest after thousands of cycles. Moreover, the proposed models are capable of estimating the probability of fault occurrence in systems exposed to temporal events of varying duration.

In order to accurately model the fault dependencies in FTs, our approach focuses on the formalization and modeling of the probabilistic behavior of FT gates and events over time. In this section, we show the modeling of temporal FT gates. The general probabilistic model (automaton) of a FT gate is expressed in Definition 1, adapted from [87].

Definition 1. *Given a TDFT gate with a set of inputs Y and an output Z , connected through a certain logic (such as AND). The priced-timed automaton (PTA) of this gate can be formally defined by a tuple $A = (L, L_0, \chi, Act, P, \mathcal{L})$, where:*

- L is a finite number of states.
- L_0 is the initial state.
- χ is a finite set of clocks.
- Act is a finite set of actions over L .
- $inv: L \rightarrow \zeta(Y)$ is an invariant condition.
- P is a probabilistic transition function $L \times \zeta(Y) \times Dist(2^Y \times L)$.
- $\mathcal{L} : L \rightarrow 2^{AP}$ is a labeling function assigning atomic propositions to different states.

In the PTA defined above, a state $(l, \nu) \in L \times \mathbb{R}_{\geq 0}^x$ is characterized such that $\nu \models inv(l)$. In any state (l, ν) , there is a nondeterministic choice of either making a discrete transition or letting time pass. A discrete transition can be made according to any $(l, g, p) \in P$, with current state l being enabled and zone g is satisfied by the current clock valuation ν . The probability of moving to location l' and resetting all clocks in Y to 0 is given by $p(Y, l')$. The option of letting time pass is available only if the invariant condition $inv(l)$ is satisfied while time elapses. Based on this definition, following we explain in detail the proposed models for each TDFT gate.

8.4.1 Proposed Probabilistic Model of the Temporal AND Gate

The probabilistic *AND* gate can be modeled as a temporal gate because of the tight dependency between the output and the inputs. As stated previously, the output is only generated if all inputs occur. However, in the case of temporal faults, input events need to happen at the same time to cause the top level fault. For example, let us imagine a component that fails in the presence of external heat (event x) AND radiation (event y). Let us assume a scenario where external heat is applied to the component for a certain amount of time, after which the heat source is dissipated. Let us also assume that the component has time to return to its regular temperature before it is affected by external radiation. In this case, since both events have happened at different points in time, the requirements for component failure have not been met.

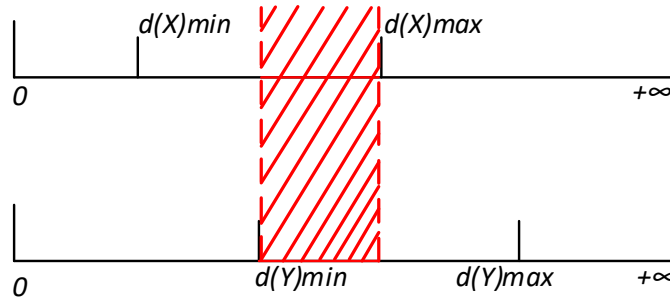


Figure 35: Possible Time Window of a TAND output

In the proposed Temporal AND gate (*TAND*), each of the basic events connected to the gate (X and Y in this example) is tied to two attributes: the probability of

the event occurring and the duration of the event. The derivation of the TAND rule is the following:

- **Duration of the Event:** It is assumed that the time duration of a basic event is a random variable d selected from an interval $[0, \dots, N]$, where $N \in \mathbb{R} \geq 0$. After an amount of time equal to d has passed, the basic event ceases to exist in the system. The basic event may re-occur according to the specified probability rate, in which case the duration of the event may be different.
- **Temporal Condition:** As is the case with regular *AND* gates, the output Z occurs only if all the inputs (X and Y) occur. However, in the *TAND* gate, the output is only generated when the duration of both inputs intersect. In other words, both inputs must be active at the same time. As illustrated in Fig. 35, the time interval in which the output of the *TAND* gate may be generated is given by:

$$Z = [d(X)_{min}, d(X)_{max}] \cap [d(Y)_{min}, d(Y)_{max}] \quad (6)$$

Where $[d(X)_{min}, d(X)_{max}]$ and $[d(Y)_{min}, d(Y)_{max}]$ are the intervals in which events X and Y are active.

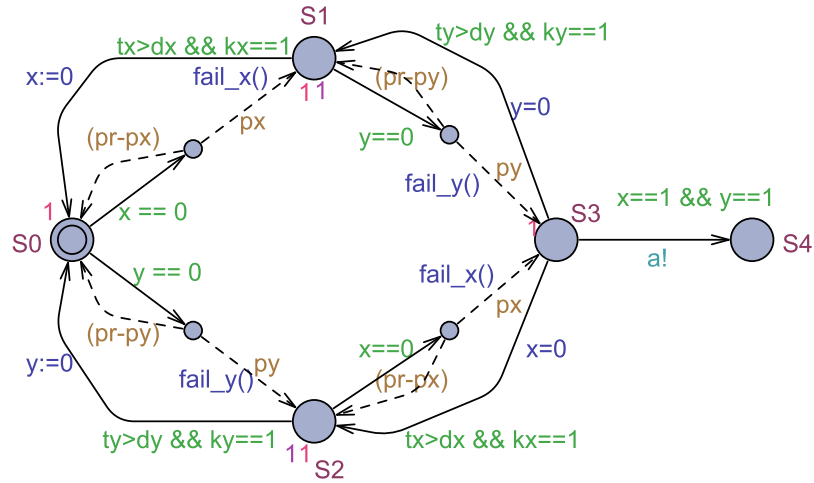


Figure 36: Example of a 2-Input Leaf TAND Gate

Fig. 36 shows a possible configuration of a *Leaf TAND* gate with two inputs. A *leaf* gate (or *bottom* gate) is any gate that has only basic events as inputs. An example of the possible flow of a *Leaf TAND* gate with two inputs is as follows: State $S0$ signifies the absence of faults. From there, the model can transition through two symmetrical paths, where either event may happen (i.e., x or y). Let us assume

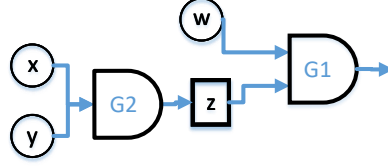


Figure 37: Example of a Simple Fault tree

that event y occurs, with probability py . In this case, the automaton moves to state $S2$ and an update is performed by the function $fail_y()$, which resets the clock ty , determines the type of the event ky ($ky == 0$ signifies permanent fault, and $ky == 1$ signifies temporal fault) and the maximum duration dy (in time units) of the event. At this point, the automaton may return to state $S0$, if $ky == 1$ and $ty > dy$, or it may proceed to state $S3$, with probability px . If event x happens, the update $fail_x()$ resets clock tx and sets the value of kx , moving the automaton to state $S3$. In this scenario, state $S3$ may transition back to state $S2$, if $tx > dx$ and $kx == 1$, or it may transition to state $S4$, broadcasting the output to the next gate. The same flow applies to the top path of the automaton, when event x occurs first.

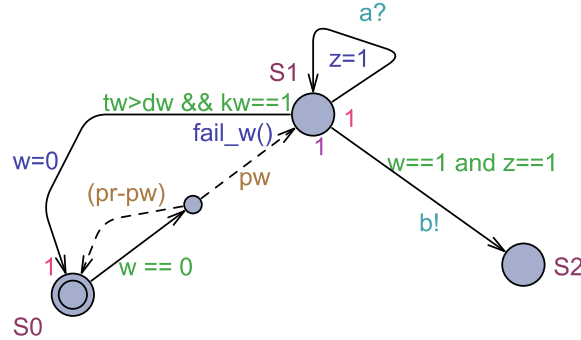


Figure 38: Example of a 2-Input Variant TAND Gate

To demonstrate how a FT is constructed with the proposed automata, a simple example is given. Let us consider the FT shown in Fig. 37. This FT has instances of two possible configurations of the 2-Input TAND gate. Gate $G2$ is a *leaf TAND* gate, since all of its inputs (x and y) are basic events. The automaton of this gate flows as described above (Fig. 36), producing output z . Let us now consider the gate $G1$ from Fig. 37. Gate $G1$ is a variant of the *leaf TAND* gate, employed in cases where one of the inputs of the gate is also the output of another gate. The PTA of gate $G1$ is shown in Fig. 38. Although gate $G1$ receives two inputs, w and z , the input z

is not an event, but rather the output of gate $G2$. Therefore, gate $G1$ only receives one basic event as input (event w). The flow of this automaton is the following: if $w == 0$ (i.e., event w is inactive), then w may happen with probability pw . If event w happens, an update is performed by the function $fail_w()$, which resets the clock tw , determines the type kw and the maximum duration dw of event w . The transition takes the system to state $S1$. In state $S1$, the automaton will wait for event z , which can be received through a synchronization channel (channel a , in this example). If the system is in state $S1$ and event z occurs (i.e., $w == 1$ and $z == 1$), then a transition to state $S2$ takes place. Alternatively, if the system is in state $S1$ and the conditions $kw == 1$ (i.e., temporal fault) and $tw > dw$ (i.e., duration of the fault has expired) are met, then the automaton goes back to state $S0$. When state $S2$ is reached, the output of the gate is produced and communicated to the next level of the FT via synchronization.

For conciseness, the discussions for the other gates will focus on the *leaf* variant, since it displays the full functionality of the gate and any required variants can be derived from the *leaf* automaton. It must also be noted that all behavioral patterns in the proposed timed automata are enforced with the use of variables, either through guards, updates, synchronization, or declarations, as exemplified above. Therefore, our use of invariants in the models has the sole goal of allowing the automaton to stay in any state indefinitely (i.e., invariant set to 1). This is important, since the propagation delay of the different inputs and gates throughout the fault tree is unknown, especially when temporal and permanent faults coexist in the same fault tree. Our experiments have shown that wrong results and often verification errors are generated when the automata are not allowed to hold a state indefinitely. Similarly, our experiments have shown that the presence of exponential exit rates is required in states where a clock is manipulated. Our results have suggested that different values of exponential exit rates have little impact on the verification, as long as the value is consistent across all states. For this reason, we have adopted the policy of setting the value of the exponential exit rate to 1 in all states where it is required.

8.4.2 Proposed Probabilistic Model of the Temporal OR Gate

The *OR* gate is modeled as a temporal gate to better represent the propagation delay that exists in real systems. The concept of the proposed leaf Temporal OR

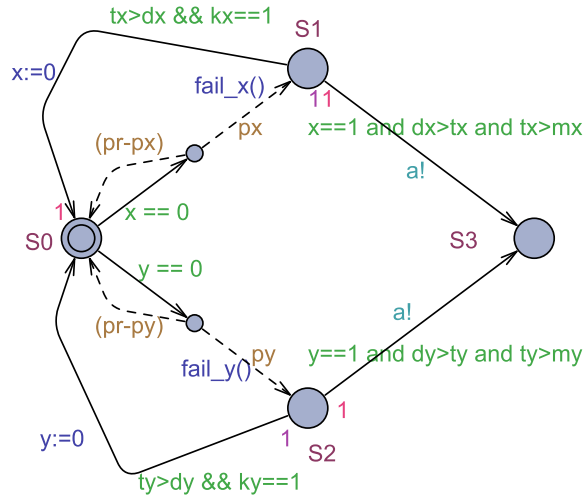


Figure 39: Example of a 2-Input Leaf TOR Gate

(*TOR*), shown in Fig. 39, is relatively simple, compared with other temporal gates: assuming two temporal events (x and y) connected to a *TOR* gate, where each event has a given probability of failure, the occurrence of either event can generate an output in the gate. However, due to the propagation delay of the modeled component (represented in Fig. 39 by the variables mx and my), the output is not generated immediately. Therefore, the output is only generated if the duration of the fault is longer than the propagation delay, but smaller than the maximum duration of the fault (variables dx and dy). The output of the gate is communicated to the next layer of the FT through a synchronization channel. The derivation of the *TOR* gate follows the same rules as the *TAND* gate with regards to the duration of the events. The temporal condition for this gate is given by the equation:

$$Z = [(d(X)_{min}, d(X)_{max}] \mid [d(Y)_{min}, d(Y)_{max}] \quad (7)$$

8.4.3 Proposed Probabilistic Model of the Temporal FDEP gate

The Functional Dependency (FDEP) gate is a dynamic gate composed of a trigger input event and one or more dependent basic events. If the trigger event occurs, the dependent events automatically become unavailable. While the trigger event is not in a failed state, the dependent events behave like regular events (i.e., the dependent events may fail independently from the trigger event). The FDEP gate does not have

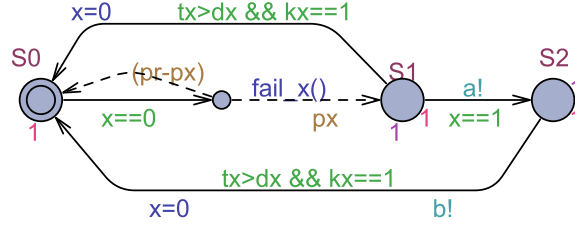


Figure 40: Example of a TFDEP Gate

a direct output, however, a functional dependency may be attached to any other gate in the fault tree, altering its behavior. The behavior of the probabilistic FDEP assumes that a dependent event y (such as the output of a gate) may happen with probability py . However, if the trigger event x occurs, with probability px , then event y is forced to happen as well.

The main issue with the regular modeling of the FDEP gate is that the temporary occurrence of the trigger event compromises the system permanently. Let us take the example of another electrical component. Let us consider the absence of electricity to power up the component as the trigger event and the failure of the component as the dependent event. It may be the case that an external interference causes the trigger event to occur, and therefore causes the component to cease function. However, it may also be the case that the trigger event disappears after a certain amount of time (i.e., the power is restored). In this scenario, the system may still function. The proposed Temporal FDEP (*TFDEP*) gate (Fig. 40), models the behavior of a temporal trigger event x . From state $S0$, event x may occur with probability px , moving the automaton to State $S1$. From state $S1$, the automaton may go back to state $S0$ if $tx > dx$ and $kx == 1$. In this case, event x has no observable effect in the FT. Alternatively, from state $S1$, the automaton may transition to state $S2$. This transition sends a message ($a!$) which immediately causes all other events that are associated to the *TFDEP* gate to fail. Furthermore, from state $S2$, if the conditions $tx > dx$ and $kx == 1$ are satisfied, the automaton may move back to state $S0$. When this transition takes place, another message is sent ($b!$), which may override the effects of the first message ($a!$). The temporal condition of an TFDEP dependency relationship is given by the equation:

$$Z = [d(X)_{min}, d(X)_{max}] \mid [d(Y)_{min}, d(Y)_{max}] \cap [d(W)_{min}, d(W)_{max}] \quad (8)$$

Where x is the trigger event, and y and w are two dependent events.

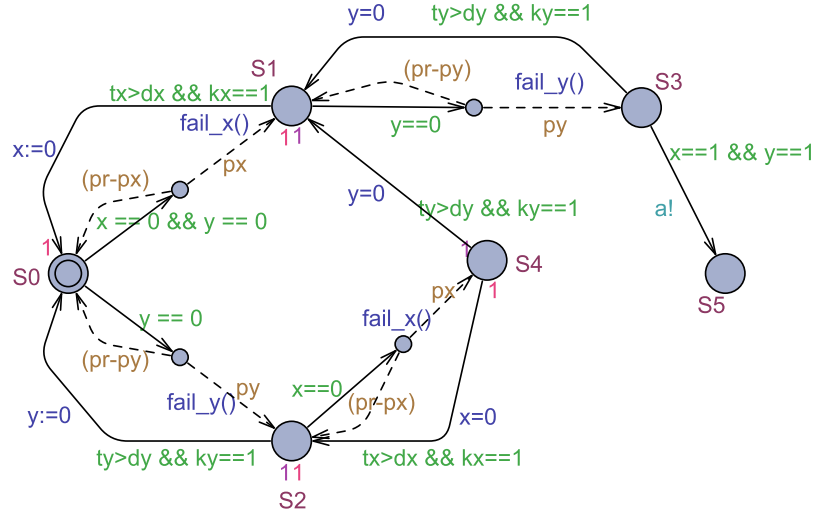


Figure 41: Example of a 2-Input Leaf TPAND Gate

8.4.4 Proposed Probabilistic Model of the Temporal PAND gate

As with the other FT gates, the PAND gate logic may also be susceptible to temporary faults. To illustrate this, let us consider the case of a backup system with two components in a standby configuration, with component A being the primary component and component B in standby. If component A fails, then the electrical switch (which can also fail) activates component B . Thus, the system will fail if component A fails and the switch fails, or if component A fails then component B fails subsequently. In this scenario, the events must occur in the specified order. However, it is possible for the switch or component B to fail without causing a system failure, if component A never fails. Moreover, it is possible that the electrical switch fails to switch due to a temporary fault. In this case, the switching action might take place immediately after the temporary fault exits the system.

In the proposed *leaf* variant of a Temporal PAND (*TPAND*) gate with two inputs (Fig. 41), starting from state $S0$ and assuming temporary event x is the primary event, a failure only happens if event x fails before event y . The failure of event x moves the automaton to state $S1$. From state $S1$, event x may disappear from the system, returning to state $S0$, or event y can happen, with probability py . The occurrence of event y transitions the automaton to state $S3$, where event y may

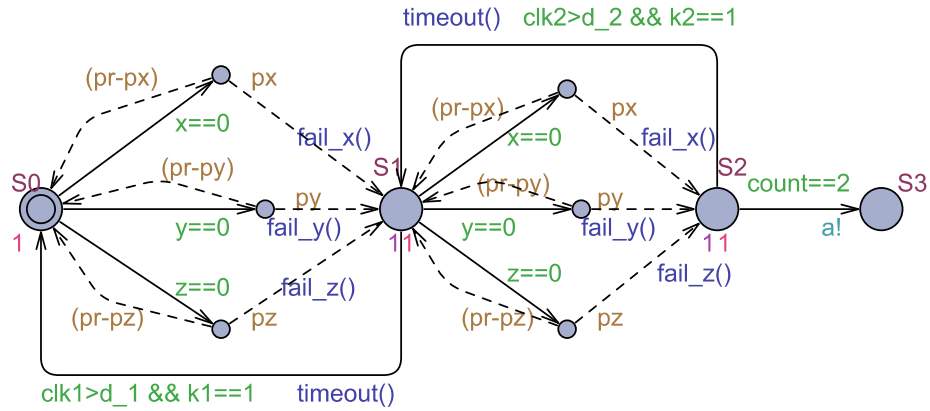


Figure 42: Example of a 2-of-3 Leaf TComb Gate

disappear (returning to state $S1$) or the output may be generated, which transitions the model to state $S5$. If event y happens first, the model moves to state $S2$. From state $S2$, event y may disappear, which causes a return to state $S0$, or event x may occur, which causes a transition to state $S4$. Finally, from state $S4$, if event y expires, the automaton moves to state $S1$. Alternatively, if event x expires, the automaton moves back to state $S2$. As with the other gates, the output of the TPAND is communicated to the next gate via synchronization message (message $a!$, for example). The temporal condition for the output of the TPAND gate is given by the equation:

$$Z = [d(X)_{min}, d(X)_{max}] \cap [d(Y)_{min}, d(Y)_{max}] \iff d(X)_{min} \leq d(Y)_{min} \quad (9)$$

8.4.5 Proposed Probabilistic Model of the Temporal COMB Gate

The Combinational gate (COMB) is a special case of the AND gate. A COMB gate is composed by three or more inputs and one output. The output occurs if M-of-N inputs occur. In other words, the combination gate allows the designer to specify the number of failures within a group of inputs that is required for the top level event to occur. By observing the examples given in the previous gates, especially the TAND and TPAND gates, it becomes clear how the combinational gate can be augmented into Temporal Combinational (*T*COMB) gate. A possible configuration of a *leaf* 2-of-3 model of the TCOMB gate is illustrated in Fig. 42. As usual, state $S0$ of the automaton signifies *no faults*. From state $S0$, three symmetrical paths may be taken,

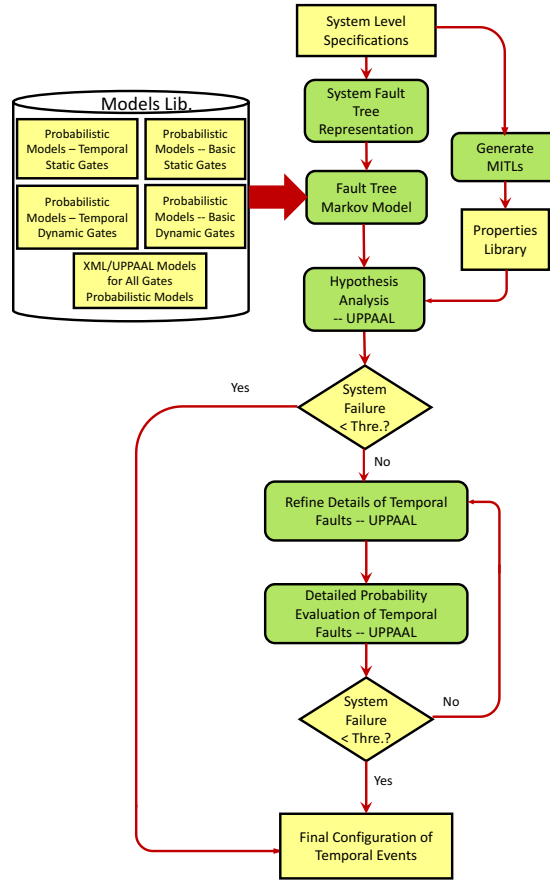


Figure 43: Main Steps of the Proposed Methodology

representing the probabilities of failure of the three inputs of the gates. Let us take event x as an example. From S_0 , event x may happen with probability px . This transition triggers the function $fail_x()$. This function is responsible for a plethora of variable updates. If the variable $count$ is equal to zero, which is the case since this is the first transition in the model, function $fail_x()$ will set the clock associated to state S_1 (clk_1) to zero, the variable d_1 will be set to the maximum duration of event x , and variable k_1 will be set to the value of kx . Finally, function $fail_x()$ will increment the variable $count$ and set variable x to 1. After all these updates, the automaton is in state S_1 . From this state, event x may expire if the conditions $clk_1 > d_1$ and $k_1 == 1$ are met. In this case, the function $timeout()$ takes place. If function $timeout()$ occurs from state S_1 (i.e., $count == 1$), all variables are reset, returning the automaton to state S_0 . Alternatively, from state S_1 , maintaining the assumption that event x has already occurred, event y can happen, with probability

py , or event z can happen, with probability pz . Let us assume that event z takes place. In this case, the automaton calls the update function $fail_z()$. Since variable $count$ is equal to one, the update function will set the clock $clk2$ to zero, the variable d_2 will be set to the value of dz , and the value of $k2$ will be set to the value of kz . The function also increments the variable $count$ and sets variable z to one. With the automaton in state $S2$ and variable $count$ equal to 2, the transition to state $S3$ may happen, broadcasting the output through channel a , in this example. However, from state $S2$, if the conditions $clk2 > d_2$ and $k2 == 1$ are met, the automaton may transition back to state $S1$, which triggers the update function $timeout()$. If $timeout()$ is called from state $S2$ (i.e., $count == 2$), the most recent event is reset (in the case of this example, z), and the variable $count$ is decremented.

The temporal condition used for the output of the TCOMB gate is a variant of the one used in the TAND gate. Therefore the temporal condition is omitted to avoid redundancy.

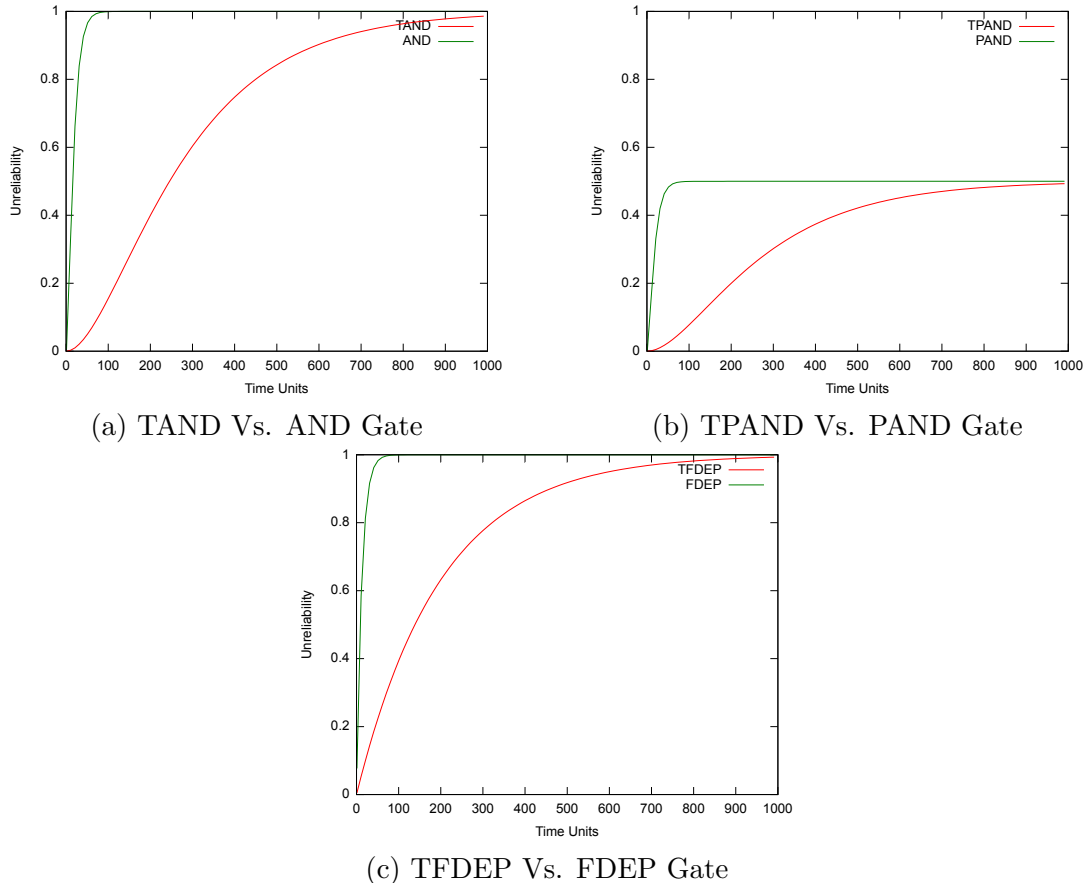


Figure 44: Comparison of the Estimated Unreliability Over Time of TDFTs and DFTs

8.4.6 Proposed Analysis Methodology

This subsection introduces the proposed analysis methodology. The flow chart of the proposed methodology is shown in Fig. 43. We start from a system-level specification model, in which a system is composed of interconnected components. This specification model must be provided by the designer in a general-purpose modeling language, such as *SysML* [61]. The failure rate of each component is characterized from the system specification. From the SysML model, a fault tree of the system is obtained using an automatic synthesis tool, such as the one proposed in [99]. Subsequently, a formal PTA model of the system’s FT is obtained through the parallel composition of the PTA models of the TDFT gates. These gate models exist in a UPPAAL gate library (shown as *Models lib.* in Fig. 43) that can be imported into any statistical model-checking tool with support to UPPAAL’s XML format. The next step is to evaluate if the reliability of the system under analysis is within the acceptable threshold defined by the specification. This evaluation is performed in UPPAAL-SMC by using Wald’s sequential hypothesis testing [141]. This test computes a proportion r among n runs that satisfy the defined property. Given two possible hypothesis, a and b , the value of r will eventually cross $\log(\beta \div (1 - \alpha))$ or $\log((1 - \beta) \div \alpha)$ with probability 1, where α and β are the probabilities of accepting hypothesis a or b , respectively [46]. The properties generated for the hypothesis testing and for the model checking steps are in the form of full weighted *Metric Interval Temporal Logic (MITL)* queries. An example of such query is given below:

$$Pr[bound; N](max : expr) \quad (10)$$

where *bound* defines the constraint on the number of runs. N gives the number of runs explicitly, and *expr* is the expression to evaluate. It is worth noting that these properties are analyzed for a certain confidence interval, which controls the number of iterations processed by the tool. If the probability of failure in the system is within the allowed threshold, no further analysis is conducted. However, if the probability of failure is above the allowed threshold, the proposed methodology moves into the refinement evaluation step. The goal of this step is to identify if there exists a certain configuration of the fault tree under analysis that satisfies the evaluated query. To this end, first, the critical path of the TDFT is identified (i.e., the sub-tree that has the highest probability of failure). Then, the leaf gates of the critical path are tested

with different instances of temporal events with different durations. For example, a gate that has only permanent faults may be changed so that one or more of its events may become temporal. Thereafter, these temporal events may be tested over different duration intervals. This analysis can generate a very a detailed quantitative report, that has the objective of showing the system designers of the exact vulnerabilities of the system, and which parameters have a greater impact to the system’s criticality.

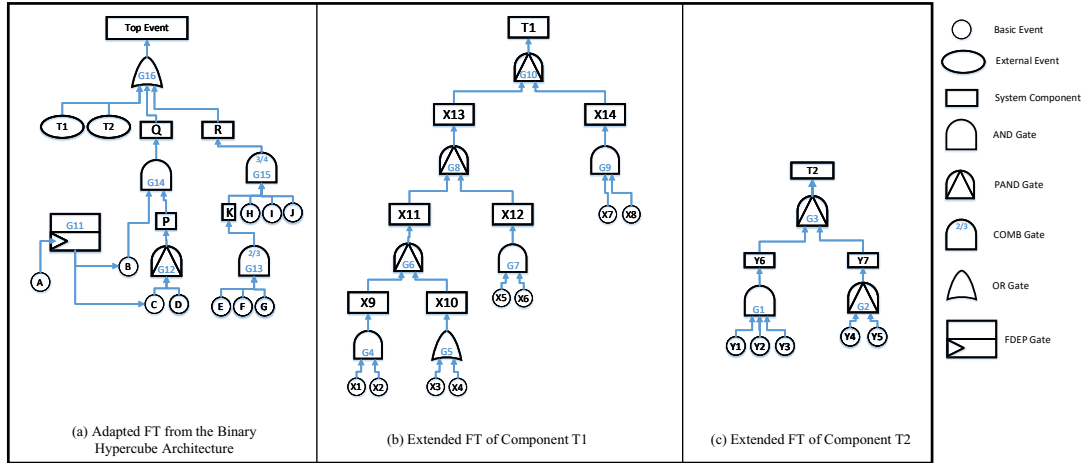


Figure 45: Modular FT Analysis of the Binary Hypercube Architecture

8.5 Experimental Results

In this section, the results of the analyses of the failure probabilities of different fault tree gates and systems with and without the presence of temporal faults are presented and discussed. Furthermore, different experiments are presented, with the purpose of demonstrating a different aspect of the proposed TDFT methodology. The analyses have been performed on UPPAAL-SMC version 4.1.19, running on a machine with an AMD Ryzen 1800X CPU and 32 GB of RAM.

8.5.1 Unreliability Evaluation Over Time

It is important to illustrate the behavioral difference between a TDFT gate and a regular FT gate. This experiment evaluates the probability of failure of a single TDFT gate due to temporal faults and the progression of this failure over time compared with a regular FT gate. Fig. 44 illustrates the behavior of different TDFT gates and their corresponding regular gates. The failure rate of all basic events in this experiment

is assumed to be equal to 0.1. Fig. 44(a) shows the results obtained from the AND gate and from the TAND gate. Fig. 44(b) shows the unreliability progression of the PAND and TPAND gates. Finally, Fig. 44(c) shows the results obtained by computing the unreliability of the FDEP and TFDEP gates, connected to a regular AND gate through a trigger relationship. It can be observed that the unreliability probabilities obtained with the regular FT gates are greatly overestimated. This happens because, as discussed previously, basic events that occur in a regular gate are permanent. However, basic events occurring in a temporal gate may have a limited duration, in which case a failure is only triggered if the required events happen during the same time window.

8.5.2 Scalability of the Proposed TDFT Analysis

One of the major weaknesses of regular FTA methods based on model-checking is the size limitation that is imposed on the analysis. For example, probabilistic model checking of FTs over time is very demanding and often cannot be handled by model-checking tools such as PRISM, MRMC and Storm. The proposed TDFT method circumvents this limitation in two distinct ways: 1) As previously mentioned (Fig. 43), the proposed TDFT analysis takes advantage of a technique called Statistical Hypothesis Testing (SHT) [141]. SHT is a method of statistical inference where two statistical data sets are compared, or a data set obtained by sampling is compared against a synthetic data set from an idealized model. In other words, the model may be evaluated against a query (i.e., is the estimated availability of component $x \geq 0.9$?). The outcome of this evaluation is either true or false. This method may be used to guide the analysis process by minimizing the expected resource consumption. 2) Unlike most model-checking techniques, the proposed TDFT methodology applies full FT modularity without the need to partition the original model into its sub-trees. This can be done by simply editing a command line, responsible for instantiating the models, in the UPPAAL system declarations.

The main objective of the second experiment is to demonstrate the applicability of the proposed approach on large systems. This experiment is also used to showcase the discrepancy between the results obtained with the proposed TDFTs against regular probabilistic FTA analysis, in scenarios where temporal faults are considered. The

Table 12: Estimated Availability of Component T1 After 100 Seconds. Failure rate of basic events is assumed to be 0.05. Temporal events are assumed to last up to 3 seconds.

	Temporal Gate	Components						
		X9	X10	X11	X12	X13	X14	T1
Regular FTA	n/a	0.0135	0.0001	0.833	0.0001	0.9023	0.0135	0.9652
TDFT	G4	0.9328	0.0001	0.997	0.0135	0.9994	0.0135	0.99997
	G7	0.0135	0.0001	0.833	0.933	0.9896	0.0135	0.99893
	G9	0.0135	0.0001	0.833	0.0135	0.902	0.932	0.998

fault trees used in this experiment can be seen in Fig. 45. Fig. 45(a) shows a top-level fault tree model that is adapted from the case study developed in Dugan et al. [50]. Components $T1$ (Fig. 45(b)) and $T2$ (Fig. 45(c)) are external events to the top-level FT. This experiment is conducted in three separate steps. In each step, two different models of the fault trees are built. The first model (DFT model) uses regular probabilistic FT gates, while the second model (proposed TDFT model) uses the temporal gates proposed in this paper. Both FT models are analyzed separately. In the case of the TDFT model, whenever an event is triggered in a temporal gate, a random choice is made to determine if the event is permanent or temporal. This is done to test the scalability of the models, as temporal events are more complex to be resolved. In the case of the DFT model, all events are considered permanent upon occurrence. It is worth mentioning that for the purposes of this paper, the results of the different iterations of the hypothesis analyses are omitted, for clarity.

Table 13: Estimated Availability of Component T2 After 100 Seconds. Failure rate of basic events is assumed to be 0.05. Temporal events are assumed to last up to 3 seconds

	Temporal Gate	Components		
		Y6	Y7	T2
Regular FTA	n/a	0.088	0.534	0.885
TDFT	G1	0.9986	0.534	0.99934
	G2	0.0879	0.9964	0.9973

The first step in this experiment is the analysis of component $T1$. The extended FT of component $T1$ is shown in Fig. 45(b). To illustrate the impact that temporal events may have in the analysis, this goal of this experiment is to consider a single temporal gate in the FT and to compute the impact of that temporal gate on the estimated availability of the system. Table 12 shows the estimated availability at the different system components when the events of gates $G4$, $G7$, or $G9$ are considered

Table 14: Estimated Availability of the Binary Hypercube System After 100 Seconds. Temporal events are assumed to last up to 3 seconds

	Temporal	Components				
	Gate	P	Q	K	R	TLE
DFTCalc	n/a	0.998	0.9993	0.996	0.99994	0.851
Regular FTA	n/a	0.998	0.9994	0.996	0.99993	0.862
TDFT	G11	0.99992	0.99996	0.996	0.99993	0.9984
	G12	0.99996	0.99995	0.996	0.99993	0.9986
	G13	0.998	0.9994	0.99998	0.99996	0.991
	G14	0.998	0.9994	0.996	0.9999991	0.9985

temporal. In this experiment, it is assumed that the failure rate of the basic events ($X1$ - $X8$) is 0.05, the duration of temporal basic events is 3 seconds, and the estimated availability is computed over a period of 100 seconds. For example, let us consider the cases where gates $G4$ or $G7$ are temporal. It can be seen in Fig. 45(b) that gate $G4$ impacts the results of components $X9$, $X11$, $X13$, and $T1$. On the other hand, gate $G7$ impacts components $X12$, $X13$, and $T1$. For each case, the results in the table quantify the impact of these gates on each of the components, compared to analyses that only consider permanent events. For example, if we were to classify the system in the terms of the *Five Nines* standard [110], and assuming that the input events of gate $G4$ are temporal, a regular FTA would classify this system (availability of $T1$) as *One Nine (1N)*, whereas the TDFT analysis would rightfully classify it as *Four Nines (4N)*. This means that in this hypothetical scenario, the regular FTA would greatly underestimate the availability rating of this system. This, in turn, would mean that the design team would have to spend more resources than necessary in order to increase the rating of the system to the *Five Nines (5N)* standard (i.e., availability = 0.999995, or the system is available for 99.999% of the time).

Next, we perform a similar analysis on the extended FT of component $T2$, shown in Fig. 45(c). For this experiment, the assumptions used for component $T1$ also apply. The results are presented in Table 13, for the regular and temporal analyses. A different analysis is performed for each of the lower-level gates to assess the impact of temporal events on those gates as well as their impact on the top event. In the last step of this experiment, we utilize the partial results in Tables 12 and 13 to analyze the fault tree of the Hypercube system (in Fig. 45(a)). The failure rates for the basic

events of this fault tree ($A - J$) are assumed to be equal to the failure rate of $T1$.

Table 14 shows an estimation of the availability of the different components of $T1$, after 100 seconds. In the table, *Regular FTA* shows the results obtained with regular FT analysis. For comparison and validation, the row *DFTCalc* of the table shows the results obtained through FTA using the DFTCalc tool [13], which is a well-known tool for dynamic fault tree analysis. In the *TDFT analysis* rows, we report the results obtained with the proposed TDFT models, with both temporal and permanent faults considered. Similarly to the previous steps of the analysis, each of the lower-level gates was analyzed individually to evaluate the effect of temporal events, as well as their impact to the top-level event. The results shown, for *Regular FTA* and *DFTCalc*, demonstrate a near parity between the values obtained. However, the table also shows that the presence of even a single source of temporal faults may drastically alter the estimated results of the analysis. This demonstrates the importance of the proposed methodology in environments where temporal faults may occur, since this difference cannot be detected with regular FTA. It is noticeable that the type and location of the temporal gate may drastically change the results obtained at the TLE. For example, gate $G11$ directly impacts the outcome of gates $G12$, $G14$ and $G16$. In other words, increasing or decreasing the availability rating associated to gate $G11$ has direct effects on the outputs of all other gates that $G11$ is connected to. Therefore, as seen in Table 14, the effects of a single gate may ripple through the FT generating significant differences in the estimated results.

Table 15: Estimated Reliability of the Pressure Chamber System After 100 Seconds. (Fault= X in the table refers to the time duration of the fault, with X being units of time.)

	(1 - Probability of Explosion)				
	Fault=1	Fault=3	Fault=5	Fault=7	Fault=10
Proposed TDFT	0.99997	0.9975	0.9911	0.9827	0.9635
TFT	0	0.69	0.69	0.69	0.69
Regular FTA	0.06	0.06	0.06	0.06	0.06

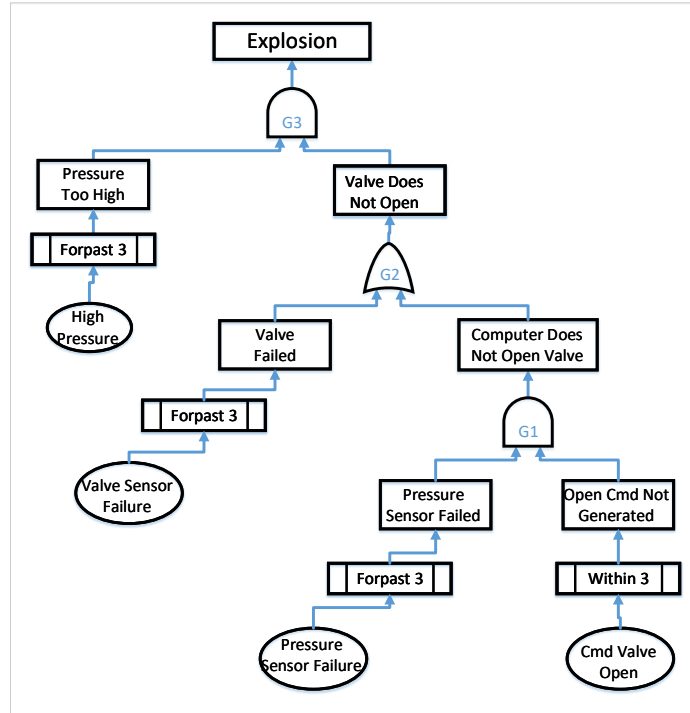


Figure 46: Temporal Fault Tree of the Pressure Chamber Case-Study

8.5.3 Comparison between TDFTs and Temporal Fault Trees (TFTs)

As previously discussed in Section 8.2, other techniques have tried to integrate temporal constraints to fault tree analysis. One of the most expressive techniques in the literature is the Temporal Fault Trees (TFTs) formalism, introduced in [108]. The analysis in [108] assigns time constraints to the propagation of the bottom events of the tree. The examples of such time constraints, shown in Fig. 46, are *Forpast* and *Within*. *Forpast* indicates that the event must be active for a minimum amount of time before propagating (e.g., *Forpast 3* indicates that the bottom event must be active for 3 units of time before propagating in the system). Similarly, *Within* indicates that the bottom event must occur within a certain time-frame (e.g, *Within 3* signifies that the event must happen before 3 time units have passed, in order to propagate in the system). In this subsection, we present a direct comparison between the TDFT and TFT techniques, by adapting the Pressure Chamber fault tree (Fig. 46) from [108] and comparing the obtained results. The system depicted in the figure shows a series of events and conditions that may lead to an explosion in the system. Starting from the bottom-most events, the failure of the *pressure sensor* for over 3

units of time together with the absence of an *open valve command* within the same 3 time units, generates an error where the computer was supposed to open the pressure valve but fails to do so (gate *G1*). Alternatively, the system may experience a *valve sensor failure*, which, if it lasts more than 3 time units, generates a *valve failure*. If the valve fails or if the computer fails to open the valve, an error is generated, since the valve did not open to release the building pressure (gate *G2*). If the valve cannot be opened and the system experiences high pressure in the chamber for over 3 time units, an explosion occurs (gate *G3*).

For the TDFT analysis, the conditions *Forpast* and *Within* have been modeled with *leaf* TOR gates, where the conditions *Forpast* and *Within* are enforced by adjusting the guards of the output transition of the *TOR* gate. Gates *G1*, *G2* and *G3* are regular TDFT gates. The comparison of the results of the TFT analysis with the proposed TDFT analysis are presented in Table 15. From the table, it can be seen that while the TFT technique is definitely an improvement over regular FTA, the results obtained by the former are rather limited. In the example (Fig. 46), the expected duration of the fault events is 3 seconds. Therefore, if the duration of the faults is less than 3 seconds, the probability of failure is equal to zero. Furthermore, if the duration of the faults is equal or greater than 3 seconds, the probability of failure is always the same. Based on this fact. The table shows that unlike other approaches, the results provided by the TDFT analysis can provide a distinct estimation for each considered fault duration.

8.5.4 Failure Estimation of the SPARC V8 Architecture with TDFTs

Having established the differences between the proposed TDFT analysis and the regular probabilistic FTA in the previous experiments, the final experiment demonstrates the importance of the proposed approach to fault analysis. This is done by analyzing the fault tree of the integer pipeline of the *Leon-3* processor and comparing the obtained results to radiation testing and simulation. In order to obtain the FT of the Leon-3 integer pipeline, we have adopted an analytical approach for the generation of fault trees for complex systems, known as the *Behaviour-Based Method* [112]. This approach considers faults as behaviours, and fault-tree gates as operations on those behaviours. By applying this technique to the Leon-3 7-stage pipeline, and based on

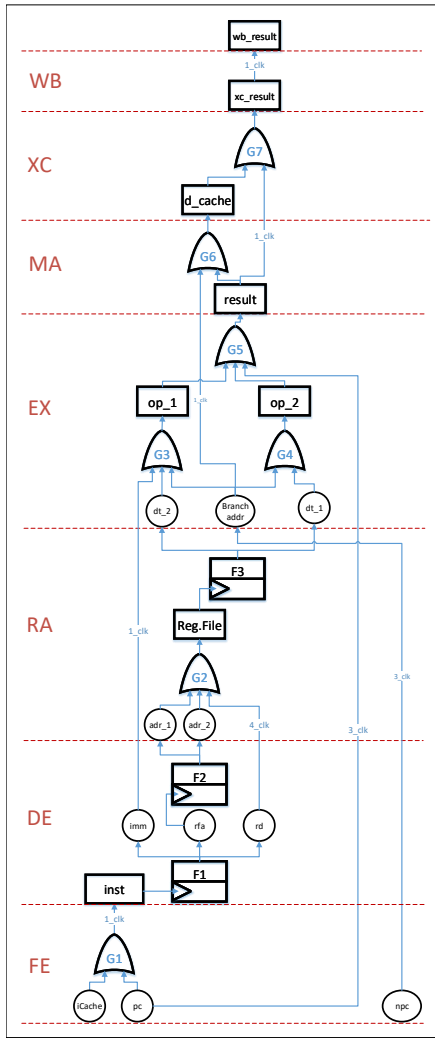


Figure 47: DFT of the 7-stage Integer Pipeline of the SPARC-V8 Architecture

the structural information available in the *SPARC V8* architecture manual [45, 74], the fault tree of the integer pipeline is constructed, as shown in Fig. 47. The fault tree is divided into 7 levels, each representing a stage of the pipeline. For this experiment, the probabilities used for the failure rates in the model are derived from the cross-section values reported in [29].

The goals of this experiment are first to determine the probability of a crash error in the processor pipeline, which raises a trap exception. Secondly, to determine the probability of each type of trap error generated over a period of time. To this end, the probabilities of soft-error events utilized in our model are derived from the cross-section values obtained through the radiation bombardment of a LEON3 design

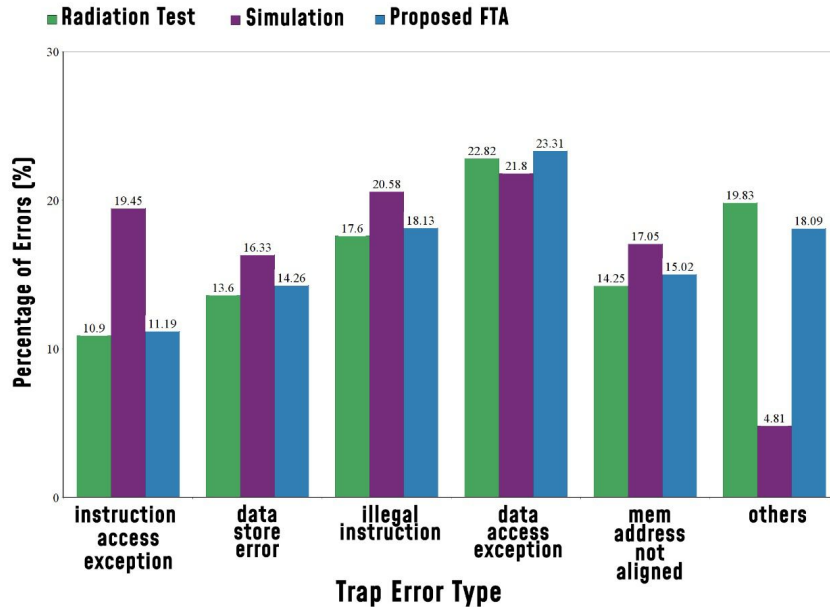


Figure 48: Probability of Trap Exceptions in different approaches. Simulation and radiation test results are reproduced from [29]

conducted and published in [29]. Furthermore, the work in [29] also contains a fault-injection simulation experiment. These results have been used in this paper for set-up and validation purposes.

In this analysis, the proposed FT model has been evaluated through over 3800 iterations, reaching a confidence level of 95%. The confidence level can be further increased if more iterations are considered. Each iteration computes the estimated probability of system failure, assuming multiple soft-errors may happen at any given time. Through our results, we estimate that approximately 41% of all errors originating from soft-faults were captured as trap exceptions. Based on the modeled fault tree, and following the functionality described in the SPARC-V8 manual [74], the proposed model is able to estimate the probability of occurrence of the following types of trap exceptions in the SPARC-V8 pipeline:

- **instruction_access_exception:** A blocking error exception causes the instruction to be unavailable.
- **data_store_error:** An error exception that occurs during a data store to memory.
- **illegal_instruction:** An attempt to execute an instruction with an invalid

Table 16: SPARC-V8 Probability of Failure Over Time

	Prob. of TE (million hours)	Prob. of Error (million hours)	Prob. of Undetected Errors (million hours)
SPARC-V8 Pipeline	0.027	0.056	0.004

opcode.

- **data_access_exception:** An error exception that occurs on a load/store data access.
- **mem_address_not_aligned:** A load/store operation that generates an improper memory address, according to the instruction.
- **Others:** All other types of trap exceptions.

Out of the 41% of soft-faults captured as trap exceptions, Fig. 48 shows the probability of each exception type to occur. In order to validate the proposed model, we have compared the obtained probabilities (*Proposed FTA*) to the ones reported in [29] (*Radiation Test* and *Simulation*). It can be seen that the values obtained with the *proposed FTA* are consistent with the values of the *radiation test*.

The proposed analysis can also be used to assess other metrics, such as the estimated time before a failure, the failure rate over time, and the impact of soft-errors on different components to the vulnerability of the system. Table 16 shows an estimation of the probability of trap exceptions over time, the probability of detected errors over time, and the probability of undetected errors over time. Although relatively small, the probability of undetected errors in the system may represent a serious issue in certain conditions, where the system is expected to operate without maintenance. Our analysis shows that the biggest contributors to the occurrence of undetected errors are the nPC register (22.5 % of cases), the rfa register (19.7 % of cases), and the d_cache (17.3 % of cases).

8.6 Conclusion

This paper presents a new modeling and analysis approach to accurately compute the availability of systems exposed to temporal faults. Regular probabilistic fault tree models calculate the probability of failure under the assumption that every fault

is permanent. However, in the real world, sources of interference (such as heat and radiation) can be intermittent. Therefore, their impact on the behavior of digital circuits (especially self-repair systems) may be only temporary. TDFTs are introduced to capture such phenomena, providing an unprecedented level of precision and customization to fault tree analysis of soft-faults. The results presented in this paper illustrate the versatility of the proposed methodology and the level of accuracy obtained in comparison with other FTA approaches. Future work includes the further extension of TDFT gates in order to analyze latent faults and rare events in systems affected by temporal faults.

Chapter 9

Conclusion and Future Work

9.1 Conclusion

This thesis set out to introduce a new method of modeling and analyzing the vulnerability of cyberphysical systems at early stages of the design cycle. As shown in the previous sections, this work is composed by two interconnected yet independent fault analysis techniques (i.e., through fault tree analysis and system-level analysis). Each of these techniques has distinct advantages and weaknesses.

From the fault tree analysis perspective, it is possible to perform a study of fault propagation which encapsulates the whole system, as complex as it may be, due to the high level of abstraction. This allows us to better understand the system's behavior as well as how that behavior can be altered in the presence of faults. This is important because understanding and predicting the most likely sources of faults in the system allows us to propose efficient design improvements to reduce the vulnerability to such faults. However, the high level of abstraction is also the weakness of fault tree analysis, since the lack of details may cause the proposed mitigation technique to be less than optimal. Another weakness of FTA is that fault trees are generated manually from the system specifications (usually written in English language), which can be a source of inaccuracies.

On the other hand, the system-level component step of the analysis is able to provide a detailed view of how faults propagate within the system, focusing on each individual component and its internal building blocks. This allows us to identify the

sources of vulnerability with a finer level of detail, which may lead to the implementation of more efficient fault mitigation solutions. The finer level of detail in this abstraction also enables us to extract fault propagation paths that may have been missed in the formulation of the fault tree, in order to enhance the system model. However, this step of the analysis is unable to provide system analysis over time due to the added complexity of the models.

This work has combined these two high-level techniques, for the first time, into one multi-level verification methodology. As demonstrated in the body of this thesis, our methodology is capable of improving the accuracy of the high-level verification process, while providing the flexibility to move across different abstraction levels. To achieve this goal, results of the more detailed component-level analysis are used to characterize the CPS sub-components. Then, these results are utilized to perform analysis of the whole system at FT-level. The combination of these approaches also improves the scalability and efficiency of the modeling as well as the verification. Namely, at each stage in the modeling hierarchy, an appropriate level of abstraction may be used to propagate the effects of errors to the next higher level. For example, the FTA step can be used to identify the critical components of a system (as a reminder, components in a fault tree are seen as black boxes). Next, the system-level analysis is performed in these components, and a detailed fault propagation graph is extracted. Finally, the fault propagation graph can be added to the fault tree of the system, replacing the black box of the component, in order to improve the accuracy of the model. In this hypothetical scenario, we are able to bypass one of the major weaknesses of FTA (the lack of details due to the high level of abstraction), without increasing the complexity of the model to prohibitive sizes. A second and more direct application of this concept is to evaluate different mitigation techniques at the different levels of abstraction, in order to find the most efficient solution. This level of iterative analysis is not attainable using conventional fault analysis techniques.

Aside from identifying the points of vulnerability in the system, one of the main challenges of the proposed methodology has been to evaluate and propose the optimal solution for the mitigation of the vulnerability. Through our work, we have identified many behavioral patterns pertaining to each fault analysis path. For example, we have identified the most common fault propagation paths in FTs, as well as the most efficient ways to decrease the vulnerability of the components in those paths. However,

in order to fully explore the vulnerability of a system, it is required to implement a connectivity channel between the different levels of abstraction. The implementation of this communication channel has been very challenging, seeing how it connects models at different levels of abstraction. In order to achieve this, we have modeled the CPS at system-level and FT-level. The fault-mitigation assessment is then conducted by comparing the results of similar mitigation techniques at multiple fault-injection points and establishing which abstraction-level produces the most efficient outcome.

Furthermore, our research has shown that the study of fault propagation in dynamic environments requires an analysis over time. For example, the effects of ionizing radiation over a circuit cannot be accurately modeled over a discrete-time model. The reason for this is that the incidence of radiation particles is a highly dynamic phenomena. Ionizing radiation may or may not be present, the flux of particles may change, the affected area of the circuit may change, and the presence of latent faults has to be considered. These events can only be accurately considered in an analysis over time. To account for this, we have proposed the concept of temporal fault tree analysis. This type of temporal FTs can support temporal and permanent probabilistic events. This analysis also provides full compatibility between FT-level and system-level models.

9.2 Future Work

Dependability analysis is one of the most important phases in the design flow of complex systems. In addition to increasing the designer's confidence in the design, an early analysis may also reduce the associated financial cost, required time and overall design effort. By introducing a versatile high-level approach, this thesis lays the foundation for a promising cross-layer approach for the early dependability analysis of CPS vulnerable to the effects of SEUs.

Since the current techniques in the literature are mostly deemed impractical and cannot be adapted for real CPS systems, future works should seek to provide practical frameworks to evaluate and improve CPS reliability and reduce the complexity of the reliability analysis while improving the accuracy of the results. These goals can be further divided into the following sub-objectives:

- **New fault models for a variety of aggressors:** Study the impact of different possible system failures classes based on the source aggressor. Different fault models should be considered for each type of aggressor in CPS, including different external types of radiation and transistor aging. Furthermore, new fault models should be proposed to characterize the impact of cyber-attacks on the reliability of the underlying telecommunication networks of the CPS. These fault models may vary according to the CPS application, criticality, environment, and design constraints. Moreover, the collective impact of different types of faults at each stage of the design cycle should be evaluated.
- **Compositional multilevel and cross-layer modeling:** Compositional verification is an important approach for verifying complex systems consisting of several interacting components. In this project, new methods to perform compositional verification should be investigated based on the concept of multilevel and cross-layer verification. In this approach, when a component is integrated into the system, it is verified against its timing and interface (i.e., interaction with other components) specifications. To achieve this, formal techniques for generating and checking both interfaces and invariants are required. Furthermore, this project should investigate analyses of unknown components at early stages of the design cycle. High-level and low-level methods may be utilized to handle detailed CPSs with different granularity, guaranteeing reliability and robustness of the system obtained by integrating incompletely specified components is a challenge for compositional verification. This should allow providing guarantees of performance and reliability validated against the requirements, while achieving acceptable cost and time-to-market objectives.
- **New formal based analysis algorithms:** Based on the target fault models, new formal based analysis solutions should be proposed, starting from the theory to real life CPS implementation. The proposed algorithm should be based on a collection of formal techniques that are suitable for each different level of abstraction. The verification of the proposed models may be achieved through the specification of suitable liveness and safety properties. The ability to ground such analysis on accurate abstraction of detailed physical implementation as well as the ability to exhaustively analyze complex systems with suitable formal methods are unheard of and will represent ground-breaking contributions.

- **Scalable architectures for complex CPS:** There is a tradeoff between the minimum number of hardware components to use and maximum reliability in a CPS. Individual components affect directly system reliability. CPS architecture should be specified with respect to software/hardware components and their inter-relationship. Therefore, system reliability must be assessed based on the reliability of components and their connections. Since cost is one of the major consideration in realizing a robust CPS architecture, ideally the component count should also be minimized. This can lead to exploiting new design choices for CPSs which are resource efficient while maintaining reliability. A set of analytical and formal tools (such as Stochastic Model Checking (SMC) and Satisfiability Modulo Theories (SMTs)) may be utilized through the coordinated execution of a prescriptive, repeatable, and measurable process.

Chapter 10

Applications

10.1 New Insights Into Soft-Faults Induced Cardiac Pacemakers Malfunctions Analyzed at System-Level Via Model Checking

Authors: Ghaith Bany Hamad, Marwan Ammar, Otmane Ait Mohamed, Yvon Savaria

Abstract: Progressive shrinking of CMOS device sizes has permitted reductions in power consumption and miniaturization of electronic devices. In parallel, modern pacemakers implemented with advanced technologies have proved to be more sensitive than earlier models to soft-errors induced notably by external radiations. Traditionally, the analysis of the impact of *soft-faults*, like those induced by Single Event Upsets (SEUs), on the behavior of pacemaker devices, has been carried out by *dynamic radiation ground testing* and *clinical observations*. However, these techniques are expensive. They can only be done very late in the design cycle, after the design is manufactured and in part after it is implanted. This paper presents a new model-based analysis of the impact of *Soft-Faults (SFs)* on the behavior of cardiac pacemakers at system-level. It is performed by: 1) introducing a new *Probabilistic Timed Automata (PTA)* model; 2) verifying this model against a set of functional properties to ensure it meets its specifications under normal conditions; 3) applying a new methodology to inject SFs at a certain time in the PTA model of the pacemaker

and to verify their impact on the pacemaker’s behavior is introduced; and 4) identifying different scenarios for *Soft-Faults (SFs)* that may lead to malfunction including *Oversensing*, *Undersensing*, and *Output Failure*. The reported formal modeling is done in PRISM and the analysis is done with the Storm model checker.

10.1.1 Introduction

The rate at which pacemakers are implanted is increasing on a global scale, with more than 700,000 new pacemakers implanted worldwide each year [102]. These devices operate at a very low voltage to reduce their power consumption and to improve their battery life. Moreover, the input nodes in a pacemaker are very sensitive, since the behavior of the system relies on capturing the intrinsic electrical signals of the heart. Therefore, the sensors of a pacemaker are very susceptible to environmental interferences. In order to achieve the high level of reliability required by these safety-critical systems, the design of the pacemaker must feature different protective measures, such as shielding in hermetic metal cases, signal filtering, inference rejection circuit, and bipolar leads [25]. With such measures, a pacemaker is protected against the everyday sources of electromagnetic radiation, such as cell phones, microwave ovens, and articles surveillance equipment, i.e., these sources are no considerable threat to the functionality of the pacemaker. However, sometimes these devices need to operate in a hostile environment (high density of radiation) that can lead to *soft-faults* which can then induce soft-errors. For these devices, there are two main environments under which they are more affected by soft-errors:

- **Radiation based treatment or exams in hospitals:** in such case, the patient is exposed to high-density external radiations. Several accounts of deaths in pacemaker patients due to Magnetic Resonance Imaging (MRI) were reported by the Food and Drug Administration (FDA) [111]. Another example of a hostile environment in hospitals is the radiotherapy treatment of cancer [150].
- **Latitude and altitude:** Soft-Faults (SFs) due to high-energy protons and neutrons vary with both latitude and altitude. For example, while traveling in an airplane, the density of high-energy protons can be 100-800 times worse than at sea-level. Electrical reset was observed during air travel due to SFs [58].

In the literature, work related to the analysis of the sensitivity of implantable cardiac devices to soft-errors is rather limited. Most of the existing techniques (such as [18, 36, 58, 71, 130, 149–151]) are based on *dynamic radiation ground testing*. This analysis provides a very accurate estimation of the vulnerability of the pacemaker. However, it is very complex, expensive, and time-consuming. Therefore, there is a growing need to analyze and estimate the impact of soft-errors due to *soft-faults* on high-level models of such systems.

For this application, the methodology proposed in this thesis is used to introduce a novel methodology to quantitatively analyze the vulnerability of pacemakers to soft-errors induced by *soft-faults* in system-level models. This work is distinct from previous works in the following ways:

- A new modeling of the behavior of the *Dual Chamber, Pacing, and Sensing* (DDD) pacemaker, at the system-level, is proposed. In this model, the behavior of each sub-component is modeled as a *Probabilistic Timed Automaton* (PTA). This is a key aspect of reliable fault analysis since probabilistic, non-deterministic behavior often arises in the presence of soft-faults and component errors. Moreover, we created a catalog of properties based on the DDD pacemaker specification and previous formal analyses of pacemakers in the literature. The correctness of the composite model is proved by verifying it against all these properties using Probabilistic Model Checking (PMC).
- A new formal probabilistic analysis of the impact of *soft-faults* on the behavior of the DDD pacemaker is proposed. The goal of this analysis is to provide full insight into the affected components, injection time, and the impact of SFs on the pacemaker behavior during each Window Of Vulnerability (WOV). This is achieved by extending the PTA of each component of the pacemaker model in order to allow fault injection and to include stochastic transitions that represent *soft-fault* propagation. The proposed technique enables quantitative investigation of SFs propagation for each injection scenario through verification of the extended pacemaker model against a set of Probabilistic Computation Tree Logic (PCTL) properties using probabilistic model checking. With this approach, different possible WOVs of the cardiac cycle have been identified. Each WOV is defined as the time interval within which a *soft-fault* at one component

of the pacemaker might impair its behavior. Then, for each WOV, we investigate the impact of a *soft-fault* on the pacemaker and on the proper behavior of the heart. If the injected *soft-fault* results into an observable soft-error, then we identify its impact on the *cardiac cycle* when it is injected and on the following cycles. Thereafter, we link the observed behaviors based on these injection scenarios with the pacemaker malfunction behaviors which were reported in the literature as part of radiation experiments or clinically in patient’s records. As it is explained later, a *soft-fault* can lead to undesirable events, such as *Oversensing*, *Undersensing*, and *Output failure* in the pacemaker.

Existing Observations of the Impact of Soft-Faults on Pacemakers

Following, we summarize the main findings in the literature on the impact of external radiations on the behavior of implantable cardiac pacemakers. For further information, the reader is referred to several reviews in the literature (such as [150], [149], [148]) that provide full details of these results. In the literature, the presence of soft-errors due to SFs in pacemakers was proven by:

1. *Clinical observations*: In these approaches, data collected from implanted pacemakers are analyzed. The reported results in [36] evaluate the incidence of SFs induced by cosmic neutron radiation in a large population of patients with cardiac implants. Other clinical observations (similar to the work done in [58]) demonstrate that the high density of cosmic radiation during air travel is linked to the electrical reset identified in the cardiac devices implanted in multiple patients. Moreover, different malfunctions were observed in pacemakers implanted in patients who suffer from cancer and have been treated by radiotherapy [71], [126]. This kind of analysis is very time consuming and only possible after pacemakers are implanted and operating in the patient’s body.
2. *Dynamic radiation ground testing*: With this approach, accelerated device testing is performed under different radiation fluxes (such as [36, 71, 130, 151]). The goal of this analysis is to replicate the observed behavior obtained from patient data and to test the vulnerability of different pacemaker models. This approach is very accurate and enables the testing of the device when exposed to a specific radiation intensity in a highly controlled environment. However,

the procedure is very expensive and only possible at post-silicon level (i.e., after the pacemaker is fully manufactured).

Both *clinical observations* and *dynamic radiation ground testing* show that the impact of SFs on the behavior of the pacemaker can be classified into three groups: 1) minor errors which do not impact the system and are only recorded in the data log of the device; 2) moderate reset, not requiring correction by the programmer; and 3) electrical reset, requiring full reprogramming of the device. Based on these results, it is evident that SFs present a real challenge to the reliability of pacemakers. However, existing analysis techniques are resource hungry, time-consuming, and require the pacemaker to be fully manufactured or even implanted.

Formal Modeling and Analysis of the Functionality of the Pacemakers

Pacemakers are safety-critical devices of which faulty behaviors can cause harm or even death. There has been much interest in developing formal verification frameworks to verify the correctness of pacemakers implementations, at different abstraction levels. Existing techniques can be classified into two categories: formal based techniques (such as [66, 75, 76, 131]) and testing based techniques (such as [77, 78]). Formal verification techniques are very efficient in providing guarantees about the pacemaker model correctness, as well as locating corner-cases and hard-to-find bugs.

In [66], the authors propose a formal specification of a pacemaker using the Z model into Perfect Developer [43]. Thereafter, based on the pacemaker specification, the correctness of the generated model was verified using the ProofPower- Z theorem prover. Tuan et. al [131] proposed the modeling of the different operating modes of a pacemaker as a Real-Time System (RTS) formal model. This model was then verified against a number of safety and correctness properties as well as timed constraints using the PAT model checker.

In the work proposed by [75, 76], a model-based framework for the automatic verification of the functionality of cardiac pacemakers was developed. The authors developed a detailed model of a basic dual-chamber pacemaker. This model is constructed based on the timed automaton [8] (TA) of each of the pacemaker's sub-components. Moreover, in this work, the authors have developed a TA of the heart behavior. The functionality of the pacemaker model has been verified using UPPAAL [24].

In [38], a quantitative functional verification algorithm for implantable pacemakers is proposed by connecting the MATLAB model of the heart, introduced in [42], and the TA model of the pacemaker in PRISM, proposed by [76]. The analysis is performed by combining both models (after exporting the PRISM model to MATLAB) and verifying the pacemaker according to its specifications.

These techniques are designed to detect bugs in the pacemaker implementation (i.e., identification of functional errors). In other words, such techniques assume that the pacemaker always operates in an error-free environment. Therefore, with these techniques, it is not possible to detect or analyze the impact of non-functional faults such as *soft-faults*. The work presented in this experiment tries to bridge the gap between high-level functional design verification and physical radiation testing, by providing a technique to perform a high-level analysis of the vulnerability of pacemakers, in hostile environments, at a very early stage of the design cycle.

10.1.2 Probabilistic Model Checking (PMC) & *Storm*

In this work, we use *Storm* [47], a powerful probabilistic symbolic model checker. It employs efficient algorithms and data structures to reduce the number of states and optimize the size of the state-machine to be solved. In addition, *Storm* supports different implementations of Markov chains, namely discrete-time and continuous-time Markov chains and Markov Automata. It also supports a wide range of probabilistic temporal logic to specify the properties to be verified such as PCTL, PCTL*, and Continuous Stochastic Logic (CSL) [19, 41]. *Storm* supports several types of input such as PRISM, JANI, GSPNs, DFTs, cpGCL, however, in this work, all the models are built in PRISM language [92]. In PRISM, a model is formed by basic constructs called modules, each designed to express a specific behavior, much like sub-components of a system. The state of each module is given by a set of finite ranged variables. The global state of the model is determined by the evaluation of the values of the module variables.

10.1.3 Steps of the Proposed Pacemaker Analysis

We propose a unified verification methodology to investigate, at the system-level, the impacts of soft-faults on the behavior of the DDD pacemaker model. As shown in

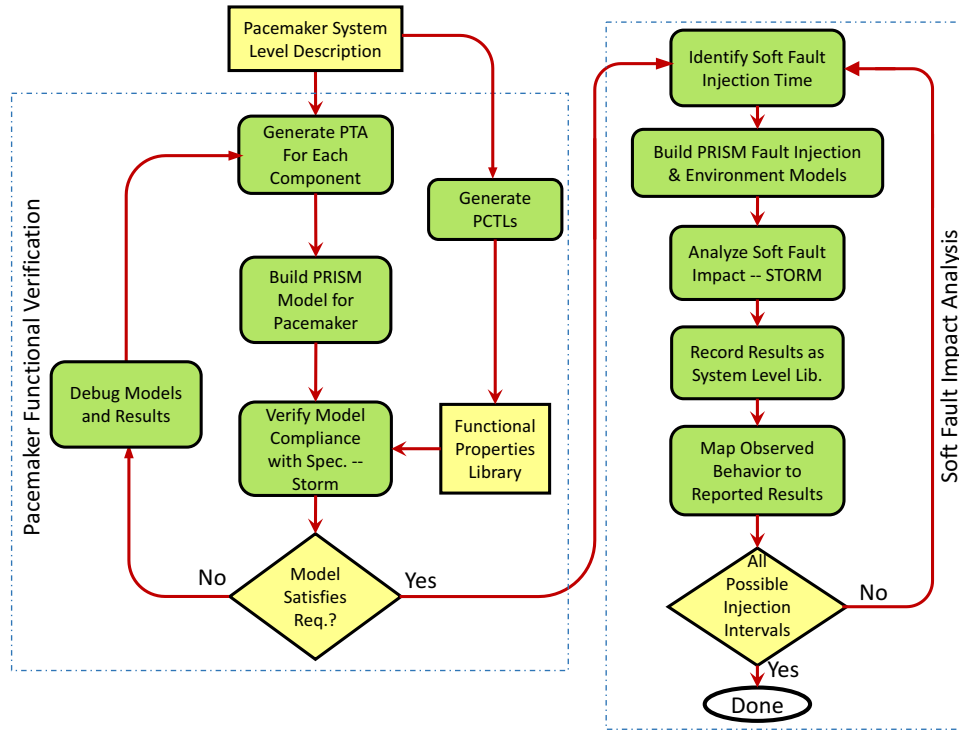


Figure 49: Main Steps of the Proposed Analysis of a Pacemaker

Fig. 49, this methodology comprises the following two main phases:

Phase 1: Model construction and functional verification: this phase starts by extracting the specification and constructing a system-level model of the pacemaker main components. Thereafter, the model of the pacemaker is constructed based on the PTA of the sub-components. This model is verified against a set of functional properties to ensure its correctness.

Phase 2: Soft-fault impact analysis: this phase operates over the model built and verified in *phase 1*. Based on the pacemaker behavior, different possible SF injection scenarios are identified. In order to analyze the impact of each SF, for each scenario, the models of the pacemaker’s sub-components and of the heart were modified to build the required fault propagation environment. The observed results are characterized and compared with the reported results from the *dynamic radiation ground testing* and/or *Clinical observations*.

10.1.4 Behavior of the DDD Pacemaker

The DDD pacemaker consists of five main components, defined as event-triggered timing cycles. The timing cycles communicate with each other through broadcasting channels, shared variables and events. In order to provide optimal hemodynamic benefit to the patient, dual-chamber pacemakers strive to mimic the normal heart rhythm. This pacemaker acts on demand, taking appropriate actions in reaction to what is happening inside the person's heart, at any given time. Following, we explain the behavior of the main components of the DDD pacemaker. In this pacemaker, the Lower Rate Interval (LRI) is the rate at which the pacemaker will pace the atrium in the absence of intrinsic atrial activity. Similar to single-chamber timing, the lower rate can be converted to a lower rate interval or the longest period of time allowed between atrial events. The LRI component is responsible for measuring the heartbeat rate and keeping it above a defined minimum value. The PTA of the LRI component is shown in Fig. 50(A). According to this figure, the LRI component monitors the ventricular sensing, ventricular pacing, and atrial sensing events. The clock of the component is reset after a ventricular event is sensed. If no atrial event is sensed within a certain amount of time, the LRI component triggers an atrial pacing (after $T_{LRI} - T_{AVI}$).

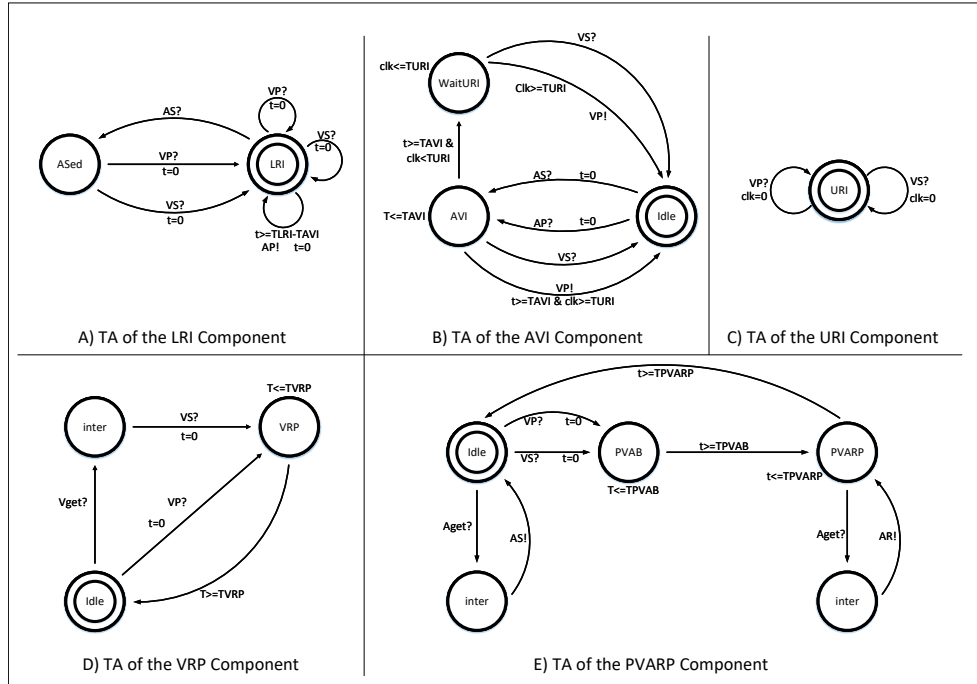


Figure 50: TAs of the Components of the DDD Pacemaker

The time that a DDD pacemaker is required to wait between a sensed or paced atrial event and a ventricular event is called the Atrio-Ventricular Interval (AVI). This behavior is implemented by the AVI component. Fig. 50(B) depicts the PTA of the expected behavior of the AVI component. As shown in this figure, the AVI component sets the longest interval between an atrial and a ventricular event. After the occurrence of an atrial event, if no ventricular event is sensed within a certain time (T_{AVI}), a ventricular pacing is performed. Correlatively, the Upper Rate Interval (URI) is defined as the upper activity rate i.e., the minimum time delay between consecutive sensor-indicated ventricle events. The URI component implements this behavior and its PTA is shown in Fig. 50(C).

The Post-Ventricular Atrial Refractory Period (PVARP) is the period of time after a ventricular pace or sense, when the atrial channel is in a refractory state. In other words, the occurrence of atrial senses during this period is identified by the pacemaker but do not initiate the A - V interval (AR). This behavior is implemented by the PVARP component, with its PTA depicted in Fig. 50(E). The purpose of the PVARP component is to avoid premature atrial contractions. This inhibits the beginning of an irregular A - V interval, which would cause the pacemaker to pace at a higher than desired rate. Similarly, the Ventricular Refractory Period (VRP) is designed to avoid restarting the V - A interval due to a noise wave. This behavior is performed by the VRP component. Fig. 50(D) depicts the PTA of this functionality. As shown in Fig. 50, ventricular sensed events, occurring in the noise sampling portion of the ventricular refractory period, are identified but will not restart the V - A interval.

10.1.5 PTA Modeling & Functional Analysis of Pacemaker

In subsection 10.1.4, the main components of the DDD pacemaker (LRI, AVI, PVARP, and VRP) are described as PTAs. As explained before, the functionality of each of these components is mainly controlled by the tight synchronization with the other components. In these PTA, the behavior of the real-time system is controlled through a finite set of *clocks* χ . The values of these clocks range over the domain $\mathbb{R}_{\geq 0}$ (i.e., non-negative real numbers). A function $\nu : \chi \rightarrow \mathbb{R}_{\geq 0}$ is referred to as a clock valuation. The set of all clock valuations is denoted by $\mathbb{R}_{\geq 0}^{\chi}$. For any $\nu \in \mathbb{R}_{\geq 0}^{\chi}$, $t \in \mathbb{R}_{\geq 0}$, and $X \subseteq \chi$, we use $\nu + t$ to denote the clock valuation which increments all clock

Table 17: Results of the verification of the functional properties of the pacemaker

Component		Verified Property	Testing Time	Result
AVI	AVI.1	$Pmax =? [F (VP = 1) \& (VS = 0)]$	$\geq T_{AVI}$	Pass
	AVI.2	$Pmax =? [F (VP = 1)]$	$\geq T_{URI}$	Pass
	AVI.3	$Pmax =? [F (VP = 1)]$	$< T_{URI}$	Pass
	AVI.4	$Pmax =? [F (VP = 1)]$	$< T_{AVI}$	Pass
	AVI.5	$Pmax =? [F (VS = 1) \& (VP = 0)]$	$\leq T_{AVI}$	Pass
	AVI.6	$Pmax =? [F (VS = 0) \& (VP = 1)]$	$\geq T_{AVI}$	Pass
LRI	LRI.1	$Pmax =? [F (AP = 1)]$	$< T_{LRI} - T_{AVI}$	Pass
	LRI.2	$Pmax =? [F (AS = 0) \& (AP = 1)]$	$\geq T_{LRI} - T_{AVI}$	Pass
	LRI.3	$Pmax =? [F (AS = 1)]$	$\leq T_{LRI} - T_{AVI}$	Pass
	LRI.4	$Pmax =? [F (AS = 1) \& (AP = 0)]$	$\leq T_{LRI} - T_{AVI}$	Pass
	LRI.5	$Pmax =? [F (AS = 0) \& (AP = 1)]$	$\geq T_{LRI} - T_{AVI}$	Pass
PVARP	PVARP.1	$Pmax =? [F (AR = 1)]$	$\leq T_{PVARP}$	Pass
	PVARP.2	$Pmax =? [F (AS = 0)]$	$\leq T_{PVARP}$	Pass
	PVARP.3	$Pmax =? [F (AS = 1)]$	$\geq T_{PVARP}$	Pass
VRP	VRP.1	$Pmax =? [F (VRP = 1) \& (VS2 = 1)]$	$\leq T_{VRP}$	Pass
	VRP.2	$Pmax =? [F (VRP = 1) \& (VP2 = 1)]$	$\leq T_{VRP}$	Pass
	VRP.3	$Pmax =? [F (VRP = 0) \& (Idle3 = 1)]$	$\geq T_{VRP}$	Pass

values in ν by t , such that $\nu(X) + t$. We use $\nu[X := 0]$ for the clock valuation in which clocks in X are reset to 0. The set of clock constrains over χ , denoted $\zeta(X)$ is defined inductively by the syntax:

$$\zeta ::= true \mid x \leq d \mid c \leq x \mid x + c \leq y + d \mid \neg \zeta \mid \zeta \wedge \zeta$$

where $x, y \in \chi$ and $c, d \in \mathbb{N}$. The clock valuation ν satisfies the clock constraint $\zeta(X)$, denoted by $\nu \models \zeta$, if and only if X resolves to true after substituting each clock $X \in \chi$ with the corresponding clock value $\nu(X)$.

Definition 2. A probabilistic timed automaton is defined by a tuple $A = (L, L_0, \chi, Act, P, \mathcal{L})$, where:

- L is a finite number of states.
- L_0 is the initial state.
- χ is a finite set of clocks.
- Act is a finite set of actions over L .

- $inv: L \rightarrow \zeta(X)$ is an invariant condition.
- P is a probabilistic transition function $L \times \zeta(X) \times \text{Dist}(2^X \times L)$.
- $\mathcal{L} : L \rightarrow 2^{AP}$ is a labelling function assigning atomic propositions to different states.

In a PTA, a state $(l, \nu) \in L \times \mathbb{R}_{\geq 0}^X$ such that $\nu \models inv(l)$. In any state (l, ν) , there is a non-deterministic choice of either making a discrete transition or letting time pass. A discrete transition can be made according to any $(l, g, p) \in P$, with current state l being enabled and zone g is satisfied by the current clock valuation ν . The probability of moving to location l' and resetting all clocks in X to 0 is given by $p(X, l')$. The option of letting time pass is available only if the invariant condition $inv(l)$ is satisfied while time elapses.

The PTA for each component is represented as a separate PRISM module. In order to accurately construct a system-level model of the pacemaker, the parallel composition of the PTAs of the subcomponents is required. The proposed methodology introduces the soft-fault (SF) injection module. This module is responsible for injecting, tracking, and synchronizing SFs across all other components, during the analysis when a fault is injected. Furthermore, the addition of the SF injection module requires significant alterations to all other modules of the pacemaker. The main challenge is to ensure that these additions and new components do not impact the core functionality of the pacemaker device, as stated in its specifications. Therefore, the proposed PTA model is verified against a catalog of functional Probabilistic Computation Tree Logic (PCTL [27]) properties obtained from an extensive review of the literature. The verified set of properties are shown in Table 17. These properties were asserted using the probabilistic model checker Storm, which provides a unique trade-off between performance and modularity, the supported solvers, and a wide range of supported modeling languages. These properties are defined based on the timing requirements of the pacemaker to verify: (i) whether the pacemaker rectifies any abnormal heart behavior by providing necessary pacing, and (ii) that the pacemaker does not induce anomalous heart behaviors by providing unnecessary pacing to the heart. We confirmed that the verified properties exhaustively verify the behavior of the pacemaker with existing analysis such as [123], [76], [38]. This analysis proves that our model correctly implements the functionality of the pacemaker. It

is important to note that, in this analysis, no faults were injected through the fault injection component. As shown in Table 17, for each component, a set of properties are verified to validate its timing requirements. The timing under which the property is verified is shown in the fourth column. The verified properties in Table 17 are the following:

- **AVI.1:** A ventricular pacing can only happen if no ventricular event is sensed within *TAVI*.
- **AVI.2:** A ventricular pacing can only happen at a time which is equal or greater than *TURI*.
- **AVI.3:** There is no reachable state where a ventricular pacing happens before the *TURI* time finished.
- **AVI.4:** There is no reachable state where a ventricular pacing happens before the *TAVI* time interval finishes.
- **AVI.5:** If a ventricular sensing happens, a ventricular pacing will not occur.
- **AVI.6:** If no ventricular sensing is detected within the expected time, a ventricular pacing will occur.
- **LRI.1:** An atrial pacing event will never take place while *test* before the *TLRI-TAVI* time interval finishes.
- **LRI.2:** If an atrial sensing is not detected within the time limit, an atrial pacing will take place.
- **LRI.3:** An atrial sensing may happen if *test* time is less or equal to *TLRI-TAVI*.
- **LRI.4:** If an atrial sensing happens, an atrial pacing will not take place.
- **LRI.5:** If an atrial sensing does not happen within the expected time, an atrial pacing will take place.
- **PVARP.1:** An atrial event happening during the time frame $t \leq TPVARP$ will be considered as an AR.
- **PVARP.2:** No atrial sensing will happen over time $t \leq TPVARP$.

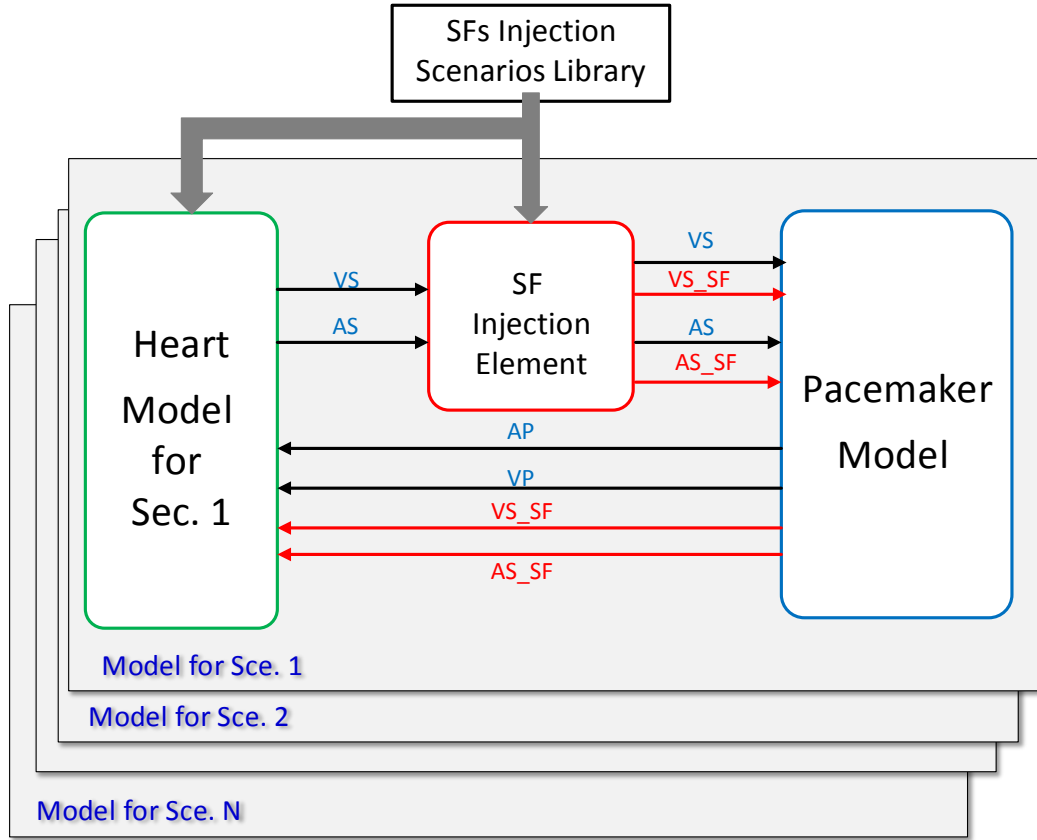


Figure 51: Proposed Formal Analysis of *Soft-Faults* Propagation

- **PVARP.3:** An atrial sensing may only happen when the time is greater than $TPVARP$.
- **VRP.1:** A ventricular sensing will not happen if time is lesser than $TVRP$.
- **VRP.2:** A ventricular pacing will not happen if time is lesser than $TVRP$.
- **VRP.3:** Component VRP will stop filtering ventricular signals when the time is greater than $TVRP$.

10.1.6 Non-functional Analysis of the Pacemaker Vulnerability to Soft-Faults

In this subsection, we present a system-level injection and analysis mechanism replicating the effects of *soft-faults* on the behavior of the pacemaker during the A-V cycle.

A new formal analysis is proposed to model and analyze each of the SF scenarios, as well as to provide new insights on the affected component, injection time, and the impact on the pacemaker behavior. The main components of the proposed analysis are shown in Fig. 51. In this analysis, the pacemaker model proposed in subsection 10.1.5 is extended to incorporate the impact of SFs.

Soft-Fault Classification

One of the main objectives of this work is to propose a vulnerability analysis of the DDD pacemaker to temporal faults at the system-level. At such high level, most of the details of the system's hardware implementation are abstracted and not yet available. Consequently, details of the sources of the vulnerability and their characteristics are not defined. In order to address these issues when modeling and analyzing temporal faults, soft-faults were introduced. A soft-fault is an abstract high-level view of any single temporal fault that can impact the pacemaker behavior for one cardiac cycle or more. Based on our review of the literature, soft-faults in pacemakers are mainly induced by ionizing radiation, which can be classified into two categories: 1) Total Ionizing Dose Effects (TIDs); and 2) Single-Event Effects (SEEs) [148], [150]. TIDs occur due to the charge accumulation in the oxide layers of the device. The oxide layers suffer from degradation if the radiation exposure is above a certain threshold (normally between 10-50 Gy (Gray units)). These levels of exposure are common in medical treatments such as radiotherapy, x-rays, and fluoroscopy. It is reported in several papers (e.g., in [148], [150]) that Implantable Cardiac Devices (ICDs) become increasingly sensitive to temporal errors over time (i.e., soft-faults). On the other hand, SEEs occur due to exposure to high concentrations of high Linear Energy Transfer (LET) particles, depositing sufficient charge to disturb normal circuit operation. Unlike total dose effects, single-event effects are ubiquitous. Thus, their significance to device reliability is of greater importance. Recent ground radiation experiments and clinical observations show that the most relevant sources of soft-faults are Single-Event Upsets (SEUs), usually originating from cosmic radiation, electromagnetic interference, or radiotherapy. Other types of single-effects are reported in the literature to have a negligible probability of occurrence [148], [150].

In the proposed modeling and analysis, *soft-faults* are introduced into the pacemaker model by adding two input nodes to the pacemaker model, namely *AS_SF* and

VS_SF. These inputs carry the SF signal i.e., it is possible to distinguish between the native (*AS* and *VS*) and faulty (*AS_SF* and *VS_SF*) events. *AS_SF* indicates the presence of an SF at the atrial input node of the pacemaker. *VS_SF* indicates the presence of an SF at the ventricular input node of the pacemaker. In our model, the pacemaker reacts to *AS_SF* and *VS_SF* as it reacts to *AS* and *VS*, respectively. As explained before, all the components of the pacemaker are synchronized based on the timing requirements of the pacemaker. It is important to note that in this analysis, it is assumed that the SFs are initiated outside the pacemaker. In other words, SFs are propagating through the main inputs (*AS*, *VS*). This can be justified due to the unavailability of a physical implementation of the pacemaker in this system-level analysis.

Formal Modeling and Analysis of Soft-Faults

In order to investigate the SF propagation, we extended the models of each component to include stochastic transitions representing SF propagation. This is achieved through the SF Injection Element (SIE), which interrupts the communication between the pacemaker and the heart. The SIE component tracks and processes the native *AS* and *VS* signals based on the desired injection scenario. Another purpose of the SIE component is to generate either *AS_SF* and *VS_SF* to derive the desired injection scenarios, as shown in Fig. 51. However, our focus is the analysis of the pacemaker model. Thus, an abstract model of the heart is used to cover only the desired behaviors. Our heart model is based on a simple synchronous communication protocol. The model is pre-programmed to release signals after a certain amount of time has passed, and to verify if the correct signals were received within a pre-defined threshold. For each scenario in our analysis, the models of the heart and of the SIE are slightly tweaked to produce the desired inputs for the pacemaker. For example, to produce the desired inputs for Sce.1, the heart model is assumed to function normally (cyclic generation of *AS* and *VS* signals). However, the SIE is programmed to eventually inject an *AS_SF* signal in the pacemaker. This injection happens after the native *VS* event, but prior to the next native *AS* event. Other scenarios may require different setup, such as Sce.4. In this scenario, the heart model is assumed to fail to produce an *AS* event, after a few cycles. When this happens, the SIE immediately injects an *AS_SF* signal in the pacemaker. Modifying the heart and SF models in

Table 18: Results of the verification of the non-functional properties related to the impact of SFs on the pacemaker

Scenario	Injection Time	Previous Events	Verification Time	Verified PCTL Property	Result	Situation illustrated in figure
Sce.1	$T_{inj} < TPVAB$	VS=1	$T_{ver} > T_{inj}$	$P_{max=?} [F (AS.SF=1)]$	1	Fig. 6
			$T_{ver} > TLRI - TAVI$	$P_{max=?} [F (AS=1)]$	1	
Sce.2	$T_{inj} > TPVAB$	VS=1	$T_{ver} < TPVARP$	$P_{max=?} [F (AS.SF=1)]$	1	Fig. 7
			$T_{ver} < TURI$	$P_{max=?} [F (AR=1)]$	1	
			$T_{ver} > TLRI - TAVI$	$P_{max=?} [F (AS=1)]$	1	
Sce.3	$T_{inj} < TURI$	VS=1	$T_{ver} < T_{inj}$	$P_{max=?} [F (VS.SF=1)]$	1	Fig. 8
			$T_{ver} > TURI$	$P_{max=?} [F (VS=1)]$	1	
			$T_{ver} < TURI$	$P_{max=?} [F (AS=1)]$	1	
Sce.4	$T_{inj} < TVRP$	VP/VS=1	$T_{ver} < T_{inj}$	$P_{max=?} [F (VS.SF=1)]$	1	Fig. 9
			$T_{ver} > TLRI - TAVI$	$P_{max=?} [F (AS=1)]$	1	
Sce.5	$T_{inj} > TPVARP$	VS=1	$T_{ver} > T_{inj}$	$P_{max=?} [F (AS.SF=1)]$	1	Fig. 10
			$T_{ver} \leq TAVI$	$P_{max=?} [F (VS=1)]$	0	
			$T_{ver} \geq TAVI$	$P_{max=?} [F (VP=1)]$	1	
Sce.6	$T_{inj} < TAVI$	AS=1	$T_{ver} < TAVI$	$P_{max=?} [F (VS.SF=1)]$	1	Fig. 11
			$T_{ver} \leq TLRI - TAVI$	$P_{max=?} [F (AS=1)]$	0	
			$T_{ver} \geq TLRI - TAVI$	$P_{max=?} [F (AP=1)]$	1	
Sce.7	$TPVARP < T_{inj} < TLRI - TAVI$	VS=1	$T_{ver} \geq T_{inj}$	$P_{max=?} [F (AS.SF=1)]$	1	Fig. 12
			$T_{ver} \leq TLRI - TAVI$	$P_{max=?} [F (AP=1)]$	0	
Sce.8	$T_{inj} \leq TAVI$	AS=1	$T_{ver} \geq T_{inj}$	$P_{max=?} [F (VS.SF=1)]$	1	Fig. 13
			$T_{ver} \geq TAVI$	$P_{max=?} [F (VP=1)]$	0	

this way greatly facilitates the conduct of the experiments.

The modeling is done by generating the PTA of each component presented in subsection 10.1.5. Each PTA is then represented as an individual PRISM module. For the purpose of our experiments, the pacemaker model is synchronized to a very simplistic heart model. Our heart model is a two-steps process that will generate one of two signals at different time delays. These time delays are defined based on the A-V cycle time delays used in [2]. The PTA of the system is obtained by the parallel composition of all PTAs of all the components of the pacemaker and the heart.

Table 18 shows the library of properties used to verify the different SF injection scenarios in different components. The second column depicts the time at which the SF is injected. The previous conditions under which the SF is injected are specified in the third column. These conditions indicate which component in the pacemaker is active and the next expected action. The fourth column shows the time in which the pacemaker model is verified against the desired properties. The properties verified in each injection scenario are reported in the fifth column. The sixth column shows the maximum probability of the verified event occurring, ranging from 0 to 1 (where 1 means 100% probability). The corresponding timing diagram of each scenario is shown in the last column of the table.

Fig. 52 illustrates an abstract view of the different SFs injection scenarios analyzed in this work. In each sub-figure, the default state is represented by two concentric

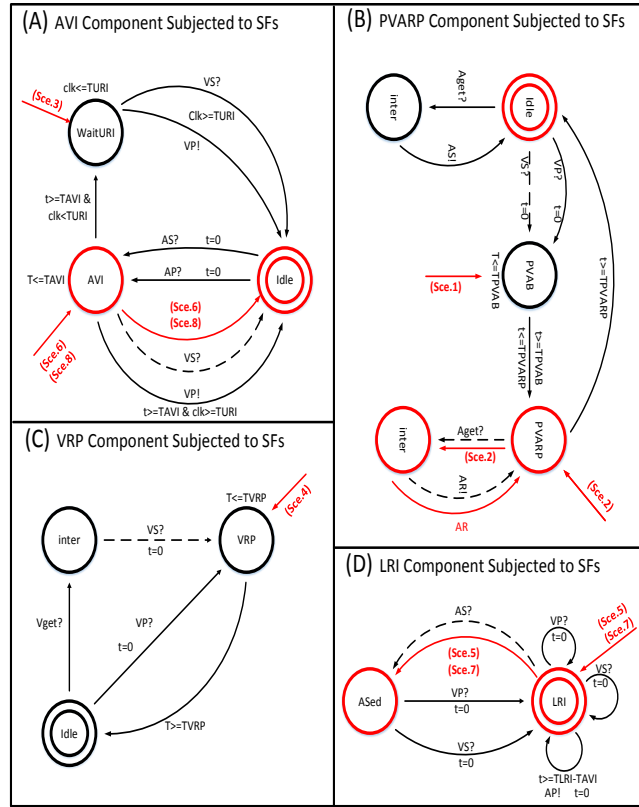


Figure 52: Effects of SFs on the Pacemaker Components

circles. The red arrows entering the components show the SF injection states in the system. Finally, the red states and transitions show which subcomponents are impacted by the SF injection. Fig. 52(A) depicts the scenario where an SF may be injected while the *AVI* component operates in the *AVI* state. The effects of such SFs are verified as shown in Table 18 (identified as *Sce.6* and *Sce.8*). Such SF may be interpreted by the system as a native VS signal. In other words, the component may erroneously identify the SF as a native ventricular activity. This event causes the state of the component to change to *idle*. However, this SF can have different effects on the behavior of the pacemaker, which are detailed in subsection 10.1.7. In the case of *Sce.6*, after the pacemaker senses an AS from the heart, the SIE injects an SF at a random time before TAVI. In this scenario, we verified three properties. With the first property, we verify that the injected SF is eventually received by the pacemaker and recognized as a native VS. The second property verifies that after the SF is received, the pacemaker resets its clock and waits for a time interval $TLRI - TAVI$, but no AS is sensed within that time interval. The third property

verifies that the pacemaker erroneously releases an atrial pacing (i.e., wrong atrial pacing due to *oversensing*). Similarly, in *Sce.8* the SIE injects an SF at a random time before TAVI, once the pacemaker senses an AS. In this scenario, we first verify that the injected SF is received by the pacemaker and recognized as a native VS. The second property verifies that after the SF is received, the pacemaker does not release the required ventricular pacing.

Another possible scenario that was analyzed is the injection of an SF in the *AVI* component at the ventricular input (VS_SF) after a native VS is sensed (i.e., during *TURI*). In this case, the *AVI* component operates in *WaitURI* state. This is shown in Table 18 and Fig. 52(A) as *Sce.3*. The pacemaker model is verified against three properties. With the first property, we verify that the incidence of SF is perceived by the pacemaker during time interval *TURI*. The second property is designed to verify that the next ventricular event is only received after time interval *TURI*. With the last property, we verify that the next atrial event (AS) is sensed within the time interval *TURI* after the native VS.

The PTA in Fig. 52(B) shows the two effects that SFs may produce on component *PVARP*, identified by *Sce.1*, and *Sce.2*. In *Sce.1*, after the pacemaker senses a native VS, the SIE randomly injects an SF at the atrial input (AS_SF) during the time interval *TPVAB* (while the *PVARP* component at state PVAB). The injection of this SF is shown in Table 18. First, we verify that the injected SF is received by the pacemaker at the specified time. The second property allows us to verify that this SF has no impact on the system. This is done by checking that the next atrial event is correctly sensed by the pacemaker (i.e., no change in the A-V timings). In *Sce.2*, the SIE injects the SF at *AS_SF2* after VS is sensed (i.e., during time interval $TPVAB < T < TPVARP$). The first two properties check if the injected SF is received by the pacemaker and characterized as a native AR event. The last property is used to verify that this SF does not affect the timing of the pacemaker. This is achieved by ensuring that the next AS event occurs within the time period $TLRI - TAVI$ as expected. Fig. 52(C) shows the effect of injecting an SF at the *VRP* component. In this scenario (*Sce.4*), after a native VS or VP event, the SIE injects an SF at a random time within the time interval *TVRP*. In order to verify this scenario, the first property is used to check that the injected SF is received by the pacemaker at the specified time. The second property verifies that the SF is

completely masked by validating that the next AS is sensed within the TLRI-TAVI.

Finally, the PTA in Fig. 52(D) shows the scenarios where an SF is injected at the LRI component, represented by *Sce.5* and *Sce.7*. In *Sce.5*, after a VS is sensed, the SIE randomly injects an SF at AS_SF within the time interval $TVARP < T_{inj} < TLRI - TAVI$. By verifying the properties for this scenario as shown in Table 18, we observed that this SF can be sensed by the pacemaker as a native AS. Next, we verify that the pacemaker component resets its internal clock and proceeds to wait for the VS signal, (i.e., the pacemaker has erroneously transitioned to the *ASed* state), breaking the A-V cycle synchronization. As shown in the result of the verification of the second property of scenario *Sce.5*, the occurrence of the SF causes the pacemaker to ignore the VS signal. In the last property, we verify that the pacemaker eventually applies an erroneous VP on the heart. In *Sce.7*, after a VS is sensed, the SIE randomly injects an SF at AS_SF. Similarly to *Sce.5*, this SF is injected within the time interval $TVARP < T_{inj} < TLRI - TAVI$. However, this SF has a different impact, which is verified as shown in Table 18. The first property confirms that this SF has been sensed as a native AS. The second property validates that such SF prevents the pacemaker from releasing the required AP.

10.1.7 New Insights on Possible Pacemaker Malfunctions Induced by Soft-Faults

In the previous sections, we introduced a system-level analysis approach designed to provide a high-level view of several possible scenarios that may lead to pacemaker malfunctions. Each of these high-level scenarios can be mapped to different occurrences of low-level faults. In this subsection, based on the results of the analysis introduced in subsection 10.1.6, and on the behavior of the pacemaker explained in subsection 10.1.4, the observed pacemaker malfunctions are mapped to physical-level analysis results reported in the literature. These reports may originate from several different sources of errors at lower-level such as SEUs and Multiple-Bit Upsets (MBUs). Based on these reports and on our analysis results, different Windows Of Vulnerabilities (WOVs) are investigated. A WOVS is defined as the time interval in which an SF at one component can impair the behavior of the pacemaker. For each WOVS in the *cardiac cycle*, if the SF results in an observed soft-error, then we identify its impact on the *cardiac cycle* where it is injected and in the future cycles. For all

these scenarios, the results of our analysis are characterized. Furthermore, we have researched the literature to identify if the observed behaviors are reported in *clinical observations* and/or as results of *dynamic radiation ground testing* of pacemakers. In the next subsections, we explain all the SFs injection scenarios, which are classified into a given subsection based on their eventual impacts. Following are some general considerations pertaining to all scenarios:

- The impact of each SF scenario (Fig. 53 to 60) is demonstrated for two cardiac cycles, as seen from the pacemaker.
- Native heart events (atrial sensing (AS) and ventricular sensing (VS)) are identified in the figures by solid black pulses, where the peaks represent atrial activity and the valleys represent ventricular activity.
- Dotted pulses represent missing or masked cardiac events or pacing events.
- Red pulses represent either SF-induced events or pacing events (atrial pacing (AP) or ventricular pacing (VP)) which are results of the injected SF.
- The “Without SF” timeline shows the expected natural progression of the cardiac cycles in the pacemaker.
- The “With SF” timeline shows the progression of the cardiac cycles in the pacemaker side after the SF has affected the system.

SF-Induced Pacemaker Oversensing

Oversensing is a phenomenon in which the pacemaker inappropriately recognizes external electrical signals and noise as native cardiac activity, and pacing is inhibited. Normally, the main sources of oversensing are large P or T waves, skeletal muscle activity, and lead contact problems. Moreover, it is reported in [140] that most common sources of electromagnetic interference (such as cellular phones) may cause pacemaker oversensing. SF-induced oversensing may lead to a disruption in the pacemaker cycle, causing undesired behavior. Through clinical tests, Hurkmans et al. [73] has reported that ionizing radiation can cause different functional inconsistencies due to sensing interference in implantable cardiac pacemakers, even leading to complete loss of function in some cases. We were able to identify the following SFs scenarios which can lead to oversensing, based on commonly adopted pacing modes, described in [115]:

1) **SF at *Aget* during TPVAB:** As explained in subsection 10.1.4, the PVARP component operates in the refractory period TPVAB to prevent the recognition of electrical signals (generated from P or T waves, skeletal muscle activity or lead contact problems) as a native cardiac activity. Additionally, an SF which is injected at input node *Aget* at time that is less than TPVAB is also considered as refractory noise. Therefore, the impact of the oversensing of such SF will be natively masked, as shown in Fig. 53.

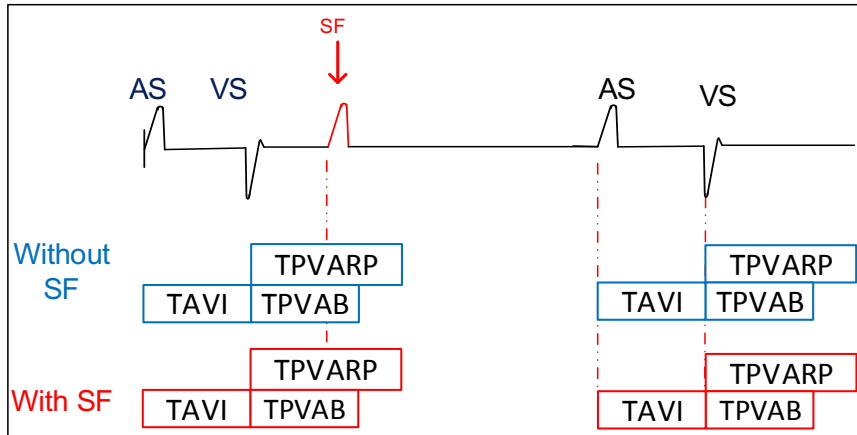


Figure 53: Timing Diagram of SF at *Aget* During TPVAB

2) **SF at *Aget* before TVARP:** An oversensing can happen in the second refractory period of the PVARP component. However, this SF will be masked if it arrives before TPVAB interval at the PVARP component. Another oversensing scenario that was considered happens if the input node *Aget* is affected by an SF during the period of time after TPVAB but before TVARP (i.e., $TAVI < T < (TLRI - TAVI)$). As shown in Fig. 54, this SF will be characterized as an *Aget* event which is then fed to the pacemaker as an *AR* event. In this case, the SF does not directly impact the pacemaker behavior, but it can impact the efficiency of the diagnosis algorithms.

3) **SF at *Vget* during TURI and TVRP:** Fig. 55 shows a scenario where multiple SFs occur during time interval $T < TURI$. In this scenario, we assume that all the SFs induce ventricular sensing signals in the system. Since the URI component limits the ventricular pacing rate in the system, all ventricular oversensing induced by SFs during TURI are masked by the pacemaker. Therefore, such SFs do not have any impact on the pacemaker behavior. Similarly, Fig. 56 shows a scenario where multiple SFs are injected during time interval $T \leq TVRP$. During this time interval, the

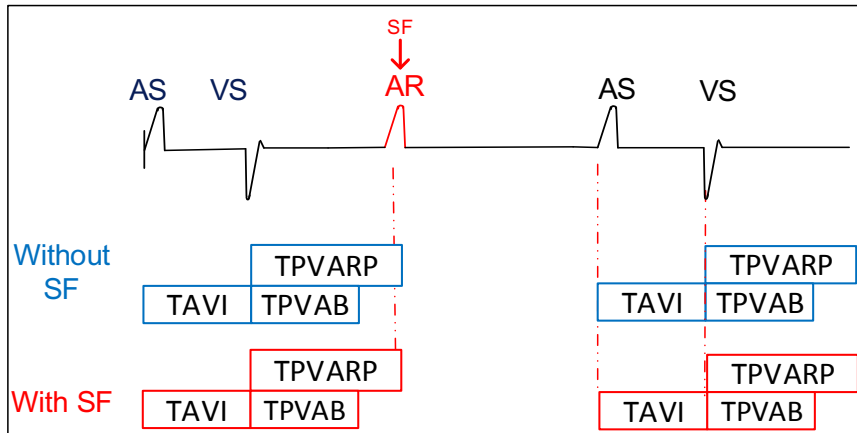


Figure 54: Timing Diagram of an SF at *Aget* During TPVARP

SFs are interpreted as refractory noise waves and are masked by the system.

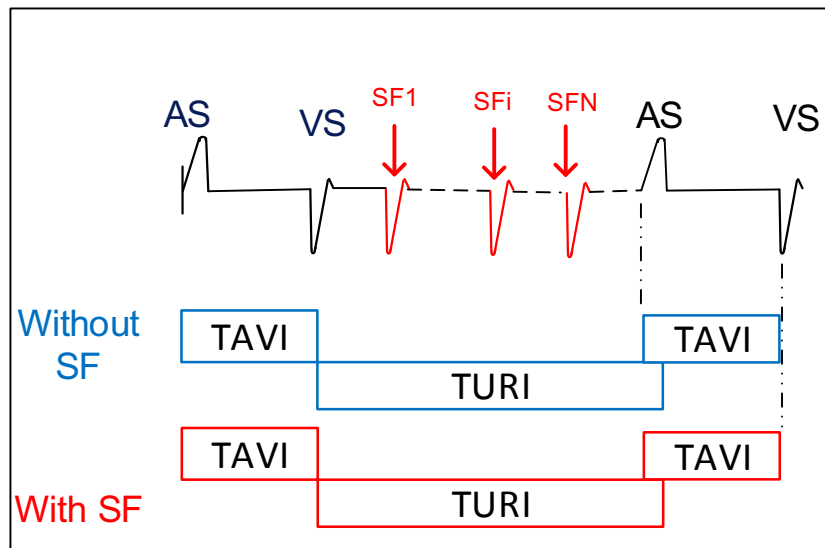


Figure 55: Timing Diagram of the Impact of SF During TURI

4) **SF at *Aget* after TVARP:** Another possible oversensing scenario is when an SF is injected at input node *Aget* at a time after TVARP. This SF can have an impact on the pacemaker behavior. This SF will be characterized as an actual *AS* event. As shown in Fig. 57, once this SF is characterized as an *AS*, then the pacemaker resets the clock and starts waiting for a *VS*, for a time interval of TAVI. This SF will have two implications: 1) it can mask the actual *AS* that is released from the

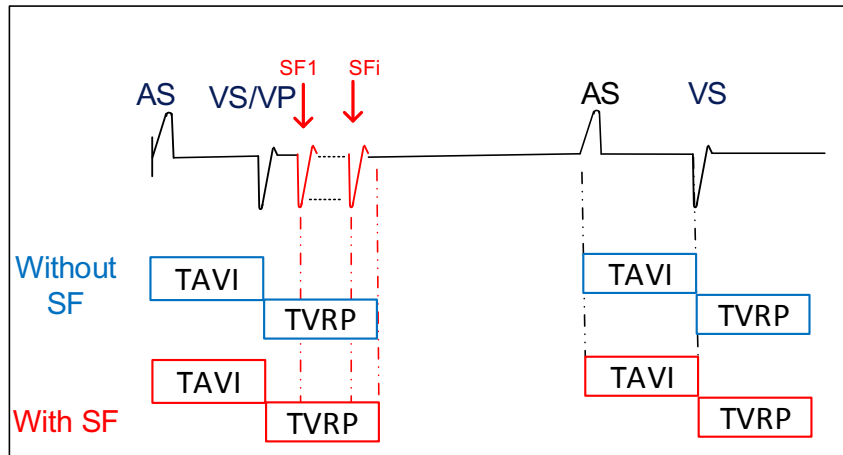


Figure 56: Timing Diagram of the Impact of SF During TVRP

heart during TAVI. Shortly after this *SF*, the real atrial event occurs but it will be filtered because the pacemaker is expecting a ventricular event after $T=TAVI$; and 2) it will impact the behavior of the pacemaker for the next A-V cycles. This happens after the pacemaker waits for the time TAVI without the heart releasing the *VS*. Therefore, the pacemaker will have to release a ventricular pacing (VP). This pacing is not required in the normal operation and will affect the *cardiac* cycle and may lead to an arrhythmia, among other issues.

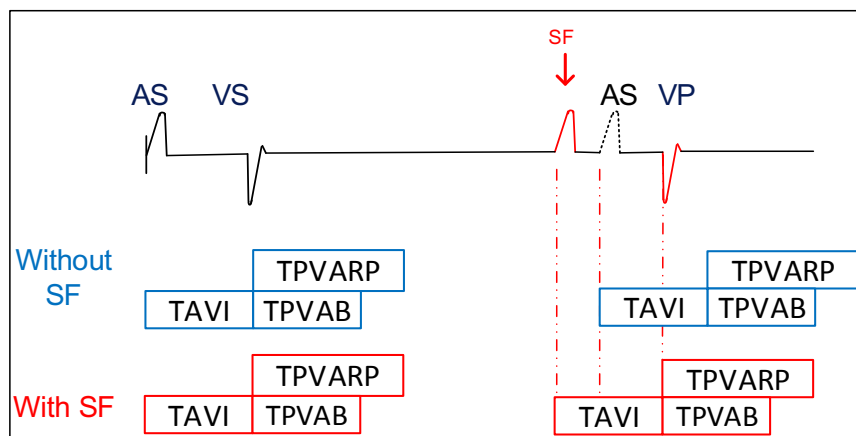


Figure 57: Timing Diagram of SF-Induced Oversensing

5) **SF at *Vget* within TAVI:** Fig. 58 depicts a possible SF injection scenario that can lead to *oversensing* at the input node *Vget* within TAVI. In this scenario, after an atrial sensing or pacing event, the pacemaker is affected by an SF which is

interpreted as a ventricular sensing before the real ventricular events happen in the heart. After the time period $T = TLRI - TAVI$, the pacemaker will be expecting to sense an atrial event. The absence of the atrial sensing, due to the fact that the system's clock is ahead of time, will cause the pacemaker to perform an erroneous atrial pacing, followed by another ventricular pacing as shown in Fig. 58.

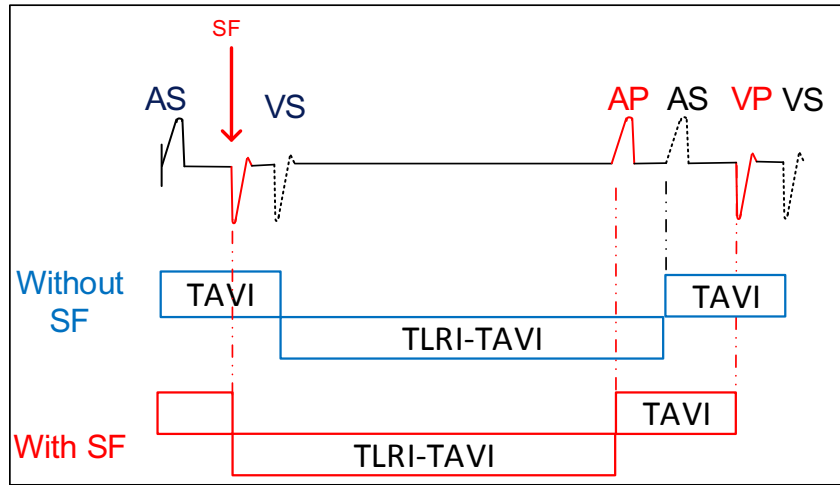


Figure 58: Timing Diagram of SF-Induced Oversensing

SF-Induced Pacemaker Undersensing

Undersensing is the failure to sense, and it occurs when the pacemaker fails to recognize spontaneous myocardial depolarization. In other words, the pacemaker fails to sense native cardiac activity. One possible scenario of an SF-induced undersensing is shown in Fig. 59. In this scenario, an SF is injected during time interval $TPVARP < T < (TLRI - TAVI)$. This SF is interpreted by the pacemaker as an atrial sensing. However, in this scenario, the SF occurrence causes the atrial activity to pass undetected. The issue is further aggravated since the pacemaker registers this SF as a AS signal and starts waiting for the ventricular event at the VS signal. After the time period $TAVI$, a ventricular pacing is erroneously applied. Therefore, the pacemaker sends an inappropriate pacing pulse to the heart. Clinically, this is normally recognized by the generation of unnecessary pacing signals and can lead to skipped beats or palpitations, among other cardiac issues [140]. Several malfunctions related to pacemaker sensing are described in [107].

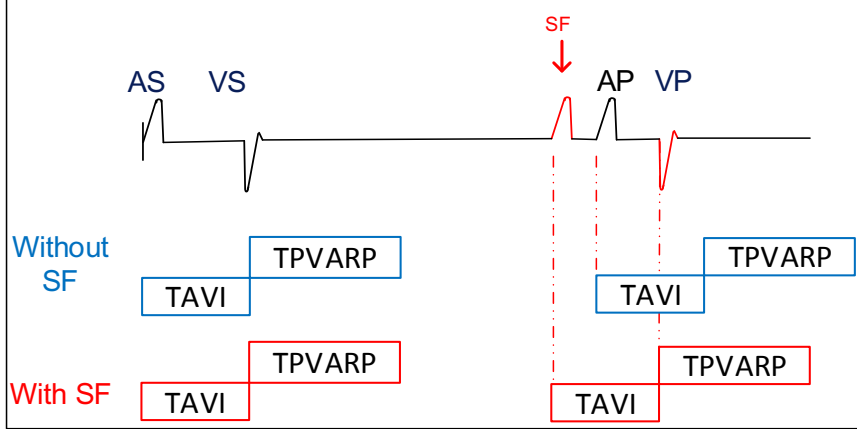


Figure 59: SF-Induced Undersensing

SF-Induced Output Failure

Output failure of a pacemaker occurs when an expected pacing stimulus is not generated. In the literature, multiple causes of output failure are identified including *oversensing* (subsection 10.1.7), pacemaker runaway, lead displacement, and electrical interference. In Fig. 60, we construct a scenario where the incidence of an SF produces an electrical signal that is interpreted by the pacemaker as an early ventricular sensing. These effects have been reported by different researchers in the literature, in works such as [73] and [115]. Output failure due to an ionizing radiation is a major concern, especially in devices with low battery charge (e.g. low battery voltage due to overdue pacemaker replacement [107]). An example of this phenomenon is shown in Fig. 60. In this example, the injected soft-fault will prevent the pacemaker from generating the necessary ventricular pacing. Furthermore, this SF will impact the pacemaker behavior in the next cycles. For instance, the pacemaker will be expecting an atrial event after time $TLRI-TAVI$ and the pacemaker performs an atrial pacing on the heart when the AS is not sensed. Thus, the pacemaker will apply the pacing to the wrong heart chamber.

10.1.8 Conclusion

In this paper, we proposed a new methodology for quantitative and automated verification of the impact of SFs on the behavior of the DDD pacemaker at the system-level. The correctness of PTA implementation is proven through model checking of a set of

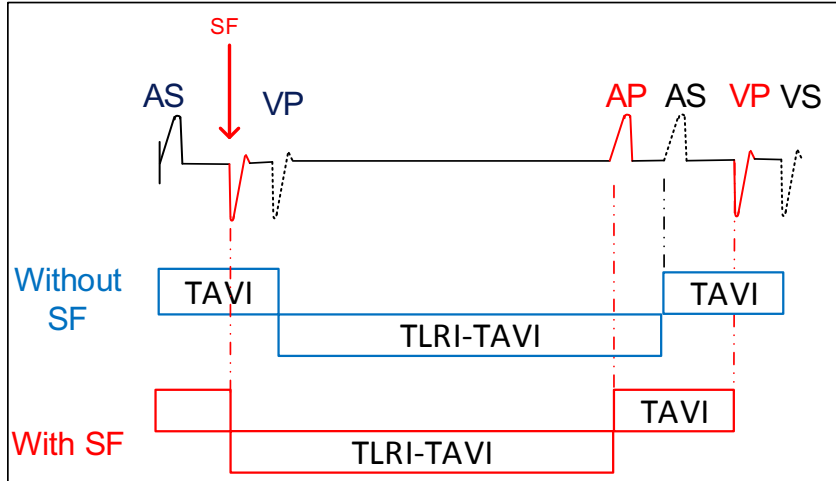


Figure 60: Output Failure due to Missed Ventricular Pacing

PCTL properties, defined based on the specifications and in agreement with the literature. We introduced a new approach to inject soft faults at certain time windows in the pacemaker model and construct an extended PTA model of the SF propagation. The proposed modeling and analysis were performed using the *Storm* probabilistic model checker. New insights on the SEU-induced malfunctions of pacemakers, such as *oversensing*, *undersensing*, and *output failure* are provided. The results of this analysis can be very useful towards improving the tolerance of the DDD pacemaker to soft-faults, by providing the necessary insight to help mitigating detected malfunctions.

10.2 System-Level Characterization, CTMDP Modeling, and Analysis of Computing Systems Reliability Applied to the LEON3 Processor

Authors: Ghaith Bany Hamad, Marwan Ammar, Otmane Ait Mohamed, Yvon Savaria

Abstract: Reliability analysis of application failures due to soft errors has become a significant concern for designers of embedded systems. The probability of these

application failures due to soft errors in microprocessors is directly related to the fault propagation path and its lifetime in the internal registers and memories. Using fault injection to evaluate the robustness of a given application program is very time-consuming, even when emulation is used, as several executions of the application are required. To improve productivity and decrease costs, the application vulnerability should be evaluated as early as possible in the development cycle. Moreover, the analysis time should be kept at the minimum. This paper presents new advances in the reliability assessment of computing systems, by introducing a new technique that addresses the combination of both software and the target hardware. To achieve this goal, we introduce a Continuous-Time Markov Decision Process (CTMDP) modeling, analysis, and estimation of the LEON3 processor’s vulnerability to Single Event Upsets (SEUs). At the system level, the proposed technique provides new insights into SEU propagation probabilities and latency. The results of the instruction based analysis are then utilized to better estimate the application vulnerability. With the proposed technique system-level insights are obtained into the application criticality presented by fault lifetime and probability. Results demonstrate that the proposed approach is able to quantify the fault propagation latency and to characterize the criticality of each of the internal registers in different applications. The results also show that the proposed approach offers a speed-up of up to 19 times over comparable techniques and is up to 1042 times faster than the best reported fault injection techniques.

10.2.1 Introduction

The progressive shrinking of devices in advanced processing technologies has made reliability one of the major concerns in the design of embedded systems [98]. Hardware systems can be affected by faults caused by physical manufacturing defects, environmental disturbances (e.g., radiations and electromagnetic interference), or aging-related phenomena. Soft-errors, induced by transient errors, are a growing issue, which degrades the reliability of embedded systems. These transient errors, such as Single Event Upsets (SEUs), can lead to undesirable changes in the state of one or more memory elements, which can propagate in the system causing soft-errors. When a soft-error reaches the software layer of the system, it can corrupt data, instructions or the control flow [105]. These errors may hinder system reliability by changing

the correct software execution or prevent the execution of an application leading to abnormal termination or application hang. Therefore, the behavior of a system in the presence of SEUs needs to be thoroughly investigated. Moreover, it is crucial to identify as early as possible the application criticality due to low-level hardware errors. The focus of this work is to investigate the vulnerability of safety-critical applications to faults affecting vulnerable components in microprocessors.

In the literature, there exist different methods to evaluate the reliability of complex systems such as the LEON3 processor, which has been widely used in automotive, multimedia systems (such as mobile phones and wireless), and other low-end and high-end applications [62]. At post-silicon stage, the reliability analysis of this system is done through *dynamic radiation ground testing* [133], [26]. In this approach, the target hardware is built and then it is exposed to different radiation fluxes based on the target analysis and system hardware. Thereafter, the number of soft-errors is counted to estimate the *dynamic cross section* (σ), which refers to the number of errors that occur in an area of a processor which executing an active load over time. *Static cross section* refers to the number of errors that occur in an area of an inactive processor.

This approach was tested on different implementations of the LEON3 processor. For instance, in [94], different experiments were performed on a LEON3 processor to evaluate the σ under different setups, such as radiating different components of LEON3 to determine which functionality has most upsets and which would require the most mitigation in a radiation environment. The main problem with this approach is that it is very expensive and time-consuming. This is mainly because it requires the system to be fully fabricated, and hence any change in the application or hardware will require a new full radiation testing to get accurate failure rate results.

In order to reduce the time and cost associated with *dynamic radiation ground testing* and provide a high-level estimation vulnerability to soft-errors of some system, different simulation and emulation fault injection methods have been proposed. These methods provide better controllability and observability than radiation testing since the user has access to the entirety of the system (injection site) and possible injection timing. In emulation fault injection, specific hardware can be built to inject these faults. For example, in [55], the *direct memory access SEU emulation* component selects the time and the bit to be altered in memory. The testing of LEON3 through

fault injection has been introduced in several works [44], [67], and [69]. Different FPGA-based frameworks are proposed in [69], to analyze the impact of SEUs on LEON3 and the expected time to recover. In [44], a platform with fault injection capability named LEON3 ViP is proposed. In these techniques, different faults are injected in the desired state elements of the LEON3 architecture, described in the HDL language. Then, the effects of these transient faults are observed by monitoring the system or comparing its outputs with a golden version of the system. However, emulation and simulation require a large amount of time to simulate or to emulate a scenario of thousands of injected faults [17]. In order to reduce the simulation time, the system is usually tested over a subset of test vectors, which reduces the results accuracy. Furthermore, these techniques require low-level implementation of the system (RTL implementation and lower). Recently, new techniques, such as [40], [105], have been introduced to analyze the expected fault lifetime in applications executed on the LEON3. These techniques succeed in reducing the analysis time over fault injection. However, along with the fault lifetime, the criticality of a fault is directly related to its propagation paths and probabilities in the application trace and in the hardware components.

Recently, formal methods have been adapted to model and analyze the impact of SEUs at the system level. Such high-level analysis provides new insights into the system reliability at an early stage of the design cycle. For example, in [11], a Continuous-Time Markov Chain (CTMC) model was proposed to compute fault propagation probabilities in a self-repairable system consisting of a small pipeline processor. This model targeted the analysis of SEUs in a three-stage pipelined unit.

We present two techniques based on fault propagation probabilities and lifetime analysis targeting both hardware and software levels of abstraction:

- The first technique (represented by *Phase 1* in the methodology) investigates faults affecting the internal pipeline registers of the LEON3 processor. This technique starts with hardware characterization to identify all the registers involved in the computation of each type of instruction. A probabilistic model of SEUs propagation probability and lifetime from each vulnerable site in the microarchitecture pipeline to the output of the instruction is proposed. To the best of our knowledge, this is the first time such CTMDP models are proposed. Based on the instruction models, the vulnerability of each type of instructions

is evaluated. This analysis identifies at what cycle instruction related information is stored in a given register, and is, therefore, able to identify the most critical registers. A new probabilistic model of SEU propagation through the LEON3 architecture is proposed. For each instruction, the propagation of SEUs is modeled as a Continuous-Time Markov Decision Process (CTMDP) based on the hardware microarchitecture of the LEON3. The proposed CTMDP models extend the work done in [57], [22], [9]. A library of CTMDP models for each instruction type in LEON3 is constructed. Based on these models, a full estimation of the fault propagation probabilities and latency through each instruction is computed.

- The second technique (represented by *Phase 2* in the methodology) investigates the lifetime of the variables of the program running on a LEON3 processor. A CTMDP model of the fault propagation through a sequence of instructions based on an application trace is proposed. This model is used to evaluate system reliability through a probabilistic approach, without the necessity of extensive fault injections. Thus, the proposed technique allows the analysis of fault propagation probabilities at the application level. This analysis provides new insights into fault lifetime expectancy.

The combination of these two techniques yields a powerful proposed system-level methodology, which is fully automated using the LLVM compiler, scripting, and Probabilistic Model Checking (PMC). The proposed methodology was validated on different case studies and for different workloads. The obtained results demonstrate that there can be a significant variance in the fault latency, register criticality and lifetime, and fault propagation probabilities between different workloads.

10.2.2 Main Steps of the Proposed Framework

The purpose of the proposed methodology is to provide a high-level estimation of the application vulnerability to SEUs. This vulnerability is estimated by evaluating two factors: 1) the failure probability due to the injected faults; 2) the fault lifetime. The main steps of the proposed method are shown in Fig. 61. The analysis is performed is done in two phases:

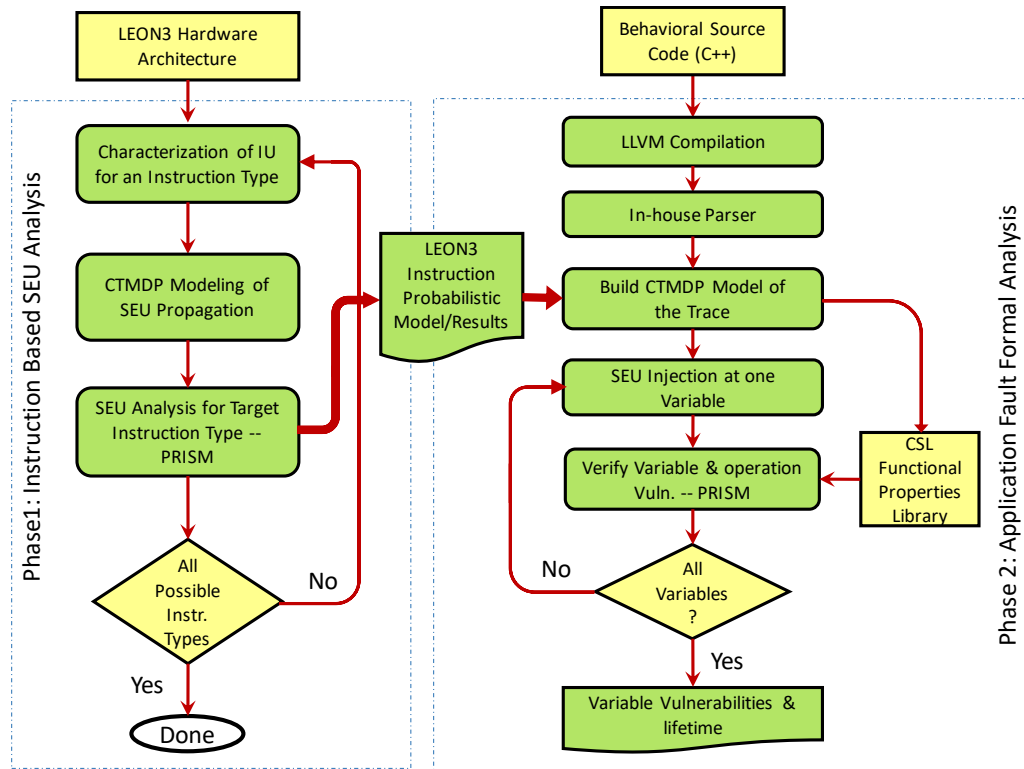


Figure 61: The main steps of the proposed methodology.

- Phase 1: Instruction based characterization, SEUs probabilistic modeling, and analysis.
- Phase 2: Application based SEUs propagation modeling and analysis.

Phase 1 starts with a high-level representation of the pipeline microarchitecture. The registers involved in the computation are identified. The instruction type is found with respect to the opcode. Each instruction type is associated with a model specifying the registers involved at each stage of the pipeline for a correct execution. This will be illustrated in subsection 10.2.3. For each cycle and pipeline stage, the instruction models are used to update a global list of "live" and "dead" registers. Next, a probabilistic model of SEUs propagation from each vulnerable site in the microarchitecture pipeline to the output of the instruction is proposed. Based on the instruction models, the vulnerability of each type of instructions is evaluated. This analysis identifies at what cycle instruction related information is stored in a given register, and is, therefore, able to identify the most critical registers. The criticality of a register is computed by summing contributions from all the cycles in which this

register is active. Details of these steps are explained in subsection 10.2.3.

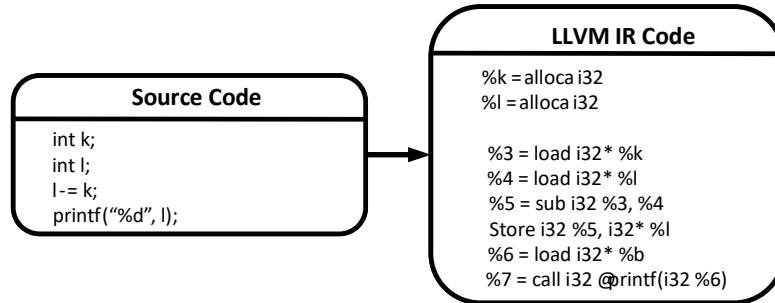


Figure 62: Example of C++ to LLVM Conversion

In *Phase 2*, the application based vulnerability analysis starts with the source code in C++. From the C++ code, we use the LLVM compiler framework to generate the intermediate representation (IR) code of the program. The IR provided by LLVM is a virtualized instruction-set and its syntax is similar to assembly language. A simple example of this process can be seen in Fig. 62. We then analyze the LLVM IR code and extract an execution trace, which includes relevant processes, such as loading from memory, storing in memory and arithmetic operations. A probabilistic modeling of the LLVM IR is proposed and automated with the PRISM language and an in-house developed parser. This model takes into account microarchitecture behaviors such as fast-forwarding. This model will be analyzed for SEUs propagation at different variables as shown in Fig. 61. In such system-level analysis, trade-offs can be made on the modeling of each instruction between accuracy and performance. The purpose of the application level analysis is to estimate the criticality of different variables during the program execution. This is achieved by estimating the lifetime of a fault that is injected in this variable and the propagation probabilities of this fault, this done through the following methods:

- **Lifetime based criticality evaluation:** This estimation is obtained by measuring the average lifetime of the variables (variables that stay alive longer tend to be more critical). In this work, the lifetime of the variable starts once a value is written to this variable and stays alive in the system until the last read operation performed in the injected fault. To achieve that, the application is first compiled into LLVM IR, however, this representation does not contain the exact timing information of the execution of each LLVM instruction. Therefore, we consider that each LLVM IR is executed within one clock cycle. This means

a counter is used to assign a time for each instruction in the target program trace. After that, the time at which the target variable is written with the fault, any time this variable is read, and the time at which this variable is overwritten are extracted. These times are used to estimate the lifetime of the target variable by evaluating the times between the write and last read. The approach of annotating the LLVM IR and the assumption that each instruction is executed within one clock cycle are adapted from literature as the work done in [40, 86, 105]. Although this assumption does not reflect reality, as different instructions have different execution times, it serves as an abstract baseline to indicate how long a variable stays alive in comparison to other variables.

- Markov reward based criticality estimation: In this method the reward is assigned to each variable on this system. If a fault is injected into any of a variable then its reward is reset, this reward is then incremented at each read operation (*load*). At the end of the analysis the reward of all variables will be evaluated to provide a measure of their criticality.

10.2.3 Instruction Based Characterization, Modeling, and Analysis

Characterization of Radiation-Induced Soft-Errors

Ionizing radiation may impact the potential on electrical nodes of micro-electronic devices, causing them to forcefully change states. The amount of ionizing particles traversing the device's sensitive area is computed based on the flux intensity of particles per square centimeter per second. If these transient radiation events transfer enough energy, then an SEU (i.e., a bit-flip in a memory cell) or a transient fault in the combinational circuitry of the device may be generated. At this stage, the SEU may propagate through different states but does not necessarily generate a system failure. In fact, most generated SEUs will be masked by different masking mechanisms. These masking mechanisms are related to SEU propagation path, system current state, and fault characteristics (e.g., injection time, site). Thus, only a subset of the faults introduced in a system will result in errors. A percentage of these errors can be detected by the system within a certain time limit, which is defined as the system's *coverage factor*. In this work, in order to accurately estimate the SEU vulnerability of different

variables in an application, details of the microarchitecture specification, technology node and SEU behavior are taken in consideration. The proposed high level analysis investigates the vulnerability of each variable by checking the SEU propagation probabilities to different states and the lifetime of the different variables. The lifetime of a variable is defined as the time period starting when an SEU is injected in a register until the time when this SEU is masked or becomes inactive (i.e., it no longer affects the system's behavior). This can be characterized by the set of states in which the variable is active, including the state where the variable is defined, every following state in which the variable is used as an operand of another operation, and all the states on the path between the definition state and the usage state [63].

Microarchitecture-Based Characterization

The proposed analysis introduces a characterization of the instruction set of a target microprocessor based on its microarchitecture specifications. The purpose of conducting this characterization is to identify the registers involved in the different steps of the required computation. The instruction type is found with respect to the opcode.

Each instruction type is associated to a model, which specifies which registers in the integer unit's pipeline are involved, during the normal execution of the target instruction, as well as different fast-forwarding cases [86]. Based on the string of instructions that needs to be executed, this approach identifies which registers are involved and the appropriate execution path for each instruction is selected. This enables an accurate high-level estimation of the lifetime of the inject faults.

For example, considering the LEON3 pipeline, the characterization of the arithmetic operation *multiplication* (MUL) includes:

- DECODE_INST register (decode stage)
- RFO_DATA1 and RFO_DATA2, RD_CTRL (Register_Access stage)
- R_E_OP1 and R_E_OP2 (Execute stage)
- R_M_RESULT (Memory access stage)
- R_X_RESULT (Exception stage)
- R_W_RESULT (Write_Back stage)

Similarly, other possible characterization of the MULimm instruction where R_A_IMM register will be active instead of RFO_DATA2. While this modeling stage is dependent on the target processor, it is a step that needs to be done only once and that can be easily adapted to a new target.

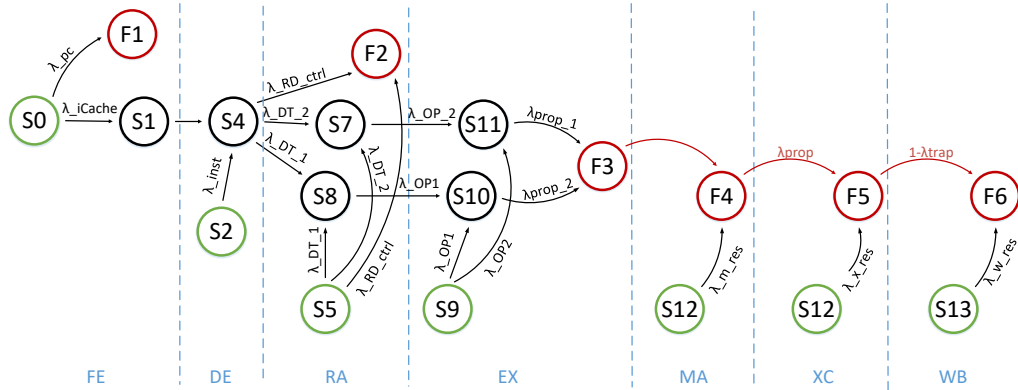


Figure 63: Proposed Probabilistic Model of SEU Propagation Through a the Processor Pipeline in an ADD Instruction

Instruction-Based Markov Modeling

The LEON3 processor implements the full SPARC V8 standard [62], including hardware multiply, divide, and multiply-accumulate instructions. SPARC is a CPU instruction set architecture derived from RISC. The pipeline of the IU of the LEON3 consists of seven pipeline stages structured according to the Harvard architecture. Based on the instruction type, with respect to the opcode, a probabilistic model of SEU propagation through the registers involved at each stage of the pipeline is proposed. This instruction-specific model includes the probabilistic details of the propagation path and the impact of an SEU on the operation/functionality of the instruction. For each type of instruction (e.g., *ADD*, *ADD_Imm*, *MUL*), a model is constructed to account for the different use of the pipeline registers. For instance, the *ADD* instruction uses different pipeline registers than the *ADD_Imm* instruction. Therefore, the possible fault propagation paths in these two instructions are not equivalent. The fault propagation paths of all considered instruction types have been obtained through the generation of counterexamples in our model-checking and modeled, in a library, as Markov chains. An illustrative example of the *ADD* instruction model is shown in Fig. 63. Starting from state *S0* (error-free fetching), at the

fetching stage an SEU affecting the PC can alter the address of the next instruction, which may result in wrong or invalid operations. This is shown in Fig. 63 in the transition from state $S0$ to state $F1$ (wrong or invalid PC) with a rate λ_{pc} , which indicates the rate of occurrence of an SEU-induced error in the PC register. Moreover, in the fetching stage, an SEU affecting the *icache* can result in an alteration in the data accessed from the memory and stored in the instruction register. This is represented in Fig. 63, in the transition from state $S0$ to state $S1$ with rate λ_{icache} , which indicates the rate of a bit-flip in the memory. In the decode phase, an SEU can affect the instruction register which is represented in Fig. 63 with the transition from state $S2$ (error-free state) to state $S4$. This transition can be done with rate λ_{inst} . Such SEU can cause an error in the decoded data for any of the operands or operation. In the register-access phase, an SEU can propagate from the decode phase to *data1* (transition from state $S4$ to state $S8$ with rate $\lambda_{DT.1}$) or *data2* (transition from state $S4$ to state $S7$ with rate $\lambda_{DT.2}$). Moreover, an SEU can propagate from the decode phase to the write address (RD_CTRL) (transition from state $S4$ to state $F2$ with a rate equal to $\lambda_{RD.ctrl}$). Moreover, an SEU can affect any of the registers at the RA phase. This is represented by the transitions from $S5$ (error-free state) to $F2$, $S7$, and $S8$ as shown in Fig. 63. In the execute stage, an SEU can propagate from the decoding phase causing execution of the wrong operation or execution of the right operation on the wrong data. This case is represented by the transitions from either $S10$, $S11$ to $F3$ with rates equal to λ_{prop2} and λ_{prop2} , respectively. In this case, the SEU may be logically masked in the data path. An error in the results of the execution phase propagates to the following stages (MA, XC, WB). Moreover, an error can be injected at the MA, XC, WB stages, resulting in an error in the stored data in the memory.

10.2.4 Fault Injection and Analysis Through CTMDP

SEUs are random events, which means that they may occur at any point or time during the execution of a program. Moreover, the effects of SEUs may branch through different areas of the system with different propagation probabilities, which further increases the complexity of the analysis. Thus, accurately modeling the effects that SEUs may have in a system is a challenging task. It has been demonstrated in our previous work that the analysis of the vulnerability to SEUs using Markov Decision

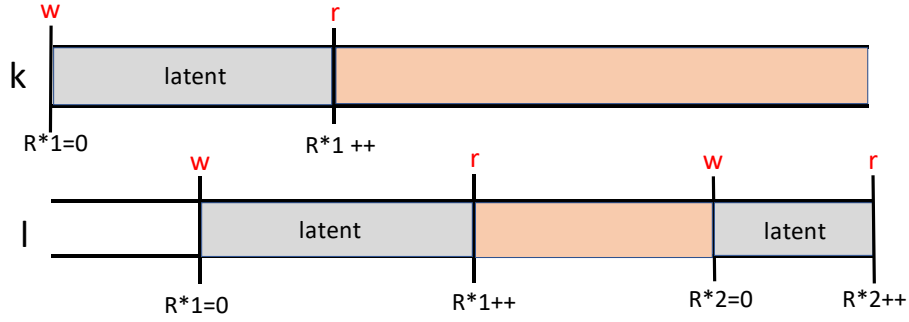


Figure 64: Example of Variable Lifetime Estimation

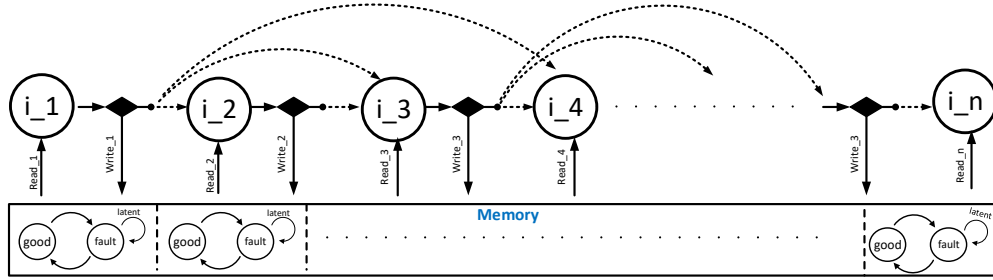


Figure 65: Probabilistic Model of SEU Propagation Through a Sequence of Instructions Based on the Target Application.

Process (MDP) [10] and CTMC [9, 11] models can offer accurate results in some cases. MDP models can capture the behavior of SEU events since they permit both probabilistic and non-deterministic choices. However, MDP models cannot be used to evaluate system vulnerability over time, since MDPs do not take into account the influence of the transition time between the states. On the other hand, CTMC models are not able to capture the non-deterministic aspect inherent to SEUs.

To address these limitations, we propose the modeling, fault injection, and analysis using the Continuous-Time Markov Decision Process (CTMDP) formalism. In comparison to MDPs, CTMDPs are able to better model the decision-making process for a system that has continuous dynamics, such as the incidence of random events over time. A CTMDP model can be defined as a tuple $\{S, S_0, (A(i), i \in S), q(j|i, a), r(i, a)\}$, where the state space S is a finite set of fully observable states of the system. S_0 is the initial state in S . $A(i)$ denotes a family of measurable subsets of actions applicable in $i \in S$. The expression $q(j|i, a)$ is the transition rate function of state j after performing action $a \in A(i)$ in state i , which can satisfy $q(j|i, a) \geq 0$

and $i \neq j$. Finally, $r(i, a) \in R$ is a reward function such that $r(i, a)$ is the immediate reward for being in state i with action a .

In this work, we utilize Probabilistic Model Checking (PMC) [19], which is a formal verification technique that can be applied to systems with stochastic behavior [59]. It does not only provide a Yes/No answer on whether a property holds, but PMC can also quantify the probability (min/max) of satisfying the property. Moreover, PMC can provide counter-examples for each particular case of property violation. The PMC tool utilized is the *PRISM* Model Checker [92], which is an efficient probabilistic symbolic model checker that employs efficient algorithms and data structures, such as Binary Decision Diagrams (BDDs). A model in PRISM is composed of several constructs called modules. Each module expresses a specific behavior, similar to sub-components in a system. The state of each module is decided by a set of finite-ranged variables, and the global state of the model is determined by the evaluation of the values of the module variables. Each module is composed of a set of commands, expressed in the format [$\langle \text{act} \rangle$] $\langle \text{guard} \rangle \rightarrow \langle \text{rate} \rangle: \langle \text{action} \rangle$, where:

- *act* is an action label used for synchronization of the different modules of the system;
- *guard* is a predicate over the operations performed in the system's modules;
- *action* is a set of n updates that will translate into operations being executed in the modules.
- *rate* is the probability of occurrence of an action;

If a guard is satisfied, the command is executed. Each command corresponds to one or more state transitions, which may be probabilistic and/or non-deterministic. Furthermore, the analysis can be enriched with the use of Markov reward structures. Reward structures extend the Markov model by incorporating a reward rate to certain actions. This mechanism, with variables that record rewards accumulated with time, enhances the analysis by providing an estimation of the most critical states in the model. Reward structures are also crucial for the lifetime estimation of SEUs. A simple example of lifetime estimation in our model is shown in Fig. 64. The figure shows two SEU injections (SEU writes) at two different variables, k and l , at different times. At the time of the injection, w , the reward count associated with each variable

is reset, and the value of the global reward count is stored. The fault stays latent in the system until the time of the first read of each variable (r). At the time of the first read, and at every subsequent read, the reward count is incremented. Eventually, if another write is performed on the variable, the value of the global reward count is stored again, and we assume that the fault has been overwritten and the value of the reward count of the variable is reset. By doing this, we can estimate the total lifetime of the fault (difference between the stored global reward values), the time between fault injection and first read, and the average time between reads. These metrics are obtained by verifying a set of Continuous Stochastic Logic (CSL [19]) properties over the CTMDP models as follows:

$$P_{S_0}(\diamond^{0,T} S_n) \quad (11)$$

Which evaluates the bounded probability of reaching state S_n within time interval $[0, T]$.

10.2.5 Application Based SEU Modeling and Analysis

In order to fully understand the real impact of an SEU at the software level, the estimation of its latency and propagation probabilities at the hardware level are essential. To evaluate these probabilities, the details of the hardware microarchitecture and the workload that is running on the processor are required. In our proposed analysis, the details pertaining to the hardware microarchitecture are included in the library of the instruction-based models, explained in Section 10.2.3. On the other hand, the details related to SEU propagation through a sequence of instructions are extracted based on the application.

This work introduces a CTMDP model of the fault propagation in an application. An abstract view of this model is shown in Fig. 65. This model is constructed based on the type of the instructions, the LEON3 pipeline microarchitecture, and the memory map. Starting from the instruction where the SEU is injected (I_1) the SEU propagation probability is evaluated to the next instructions based on their dependencies. In order to model the different paths for fast forwarding mechanism, after each instruction, a decision is made on the dependencies. If the forwarding is not needed then the faulty result is written in the register file or memory. In this case, the following instructions will be affected by this fault only, if they use the corrupted

data from the memory/register file before it is over-written. If forwarding is required then the faulty result is forwarded to the next instructions. For example, as shown in Fig. 65, if a fault at I_1 can then be forwarded to I_2 , I_3 , or I_4 . On the other hand, the proposed model takes into consideration the impact of SEUs injected into the memory. This model is based on the probability that an SEU can change the data in the memory. If this is the case, the target probabilistic model of this address moves to the *fault* state (indicating wrong data). This model of the memory address will move to the error-free state (*good*) when another instruction which is error free overwrite this address. During this period the fault is considered as latent. Before the faulty data is overwritten, any other instructions reading this address will be affected by the fault.

In *Phase 2*, similar to *Phase 1*, we utilize the PRISM model checker to automate model analysis. Each instruction is expressed as PRISM modules to specify the behavior explained before. The state of each module is decided by a set of finite-ranged variables which are controlled by the flow of the LLVM instructions defining the application. In each instruction, based on the fault flow, the probability of the fault propagation to the output of this instruction is evaluated. Moreover, the lifetime of the propagating fault is updated and traced based on Markov reward structures. In this model, lifetime estimation starts with the instruction where the fault is injected. The value of the global reward variable is recorded and incremented for the transitions where the fault is active and not flushed from the system. In the analysis stage, the reward value is used to determine the relative amount of time that the SEU was alive in the application. Similar to the instruction based analysis, the application based analysis is conducted by verifying a set of CSLs properties over the CTMDP models. An example of such CSL property is given below:

$$P_{I_1}(\diamond^{0,T} I_n) \tag{12}$$

This expression evaluates a bound on the probability that a fault, that is injected at instruction I_1 , to eventually reach instruction I_n within time interval $[0, T]$.

10.2.6 Experimental Results

In this subsection, we present and discuss the results of the proposed analyses which have been performed on a machine with an Intel Core I5-4200U CPU and 8 GB of

RAM. The instruction and application-based CTMDP models explained previously are formulated and analyzed in PRISM 4.4 [92].

The average size of the formal models is around 7.6 million states and the average model construction time is around 8.1 seconds. It is assumed that all the registers and the memory are exposed to a flux of radiation. However, different bits may have different bit-flip probabilities. The proposed analysis consists of identifying unintended bit-flips and measuring their impact on system reliability. In order to obtain an accurate verification environment, we utilize the results reported in [68] to study the impact of bit-flips in different workload configurations. It is important to note that the number of instructions in a LEON3 workload is prohibitively large (in the order of tens of hundreds of millions of instructions), and a full trace of the program execution is rather useless for the problem being investigated (i.e., fault propagation and latency). Therefore, in order to efficiently estimate these metrics, we analyze a representative subset of each workload execution trace.

As an example of such analysis, the first experiment investigates the impact of different SEU injections in different LEON3 workloads. In this experiment, it is assumed that the bit-flip rate in all the bits is the same, however, the probability of SEU in a register may vary according to how many bits are used in the instruction. For instance, in the NOP instruction, only 5 out of 32 bits are used. Table 19 shows the failure propagation rates in different workloads. It is observed that different workloads have different SEU propagation characteristics (i.e., error rates). This happens because each workload applies its own algorithm for memory mapping and memory access. Therefore, different workloads are expected to have different fault propagation and fault latency metrics. It is observed that workloads which have similar memory mapping algorithms are expected to have similar fault propagation behaviors (i.e., latency and memory mapping). For example, the percentage of latent errors is much higher in the AES workload. This happens because the memory allocation algorithm of AES adopts a progressive-write method, which means an overwrite is less likely to happen. Therefore, a fault in the memory will remain latent for a longer time. This analysis consumed 477.9 Kb of memory and took 7.4 seconds to perform.

Following up on the results discussed previously, the second experiment consisted in computing the average fault propagation latency in the different workloads. This

Table 19: Application Based Analysis of Fault Propagation

Workload Subset	Rate of Masked Errors	Rate of Failure	Rate of Latent Faults
QSort	70.5	12.78	17.22
CRC32	72.2	17.4	10.4
AES	27.66	43.2	29.14

was done by estimating the average fault propagation latency inside the integer unit (i.e., from the moment of the occurrence of the SEU in the registers until the moment when the SEU is written in the memory). The average fault latency in memory is also evaluated. This latency is computed from the time when the SEU is written in the memory until it is overwritten. These results were obtained with the use of a Markov reward model and the standard probabilistic distribution of CTMDPs. To calculate the passing of time, it is assumed that every state transition in the model corresponds to a one-time unit. This approach is not able to determine the number of clock cycles that the fault remains in the system, but it provides a quantitative baseline in which the studied workloads may be evaluated. The results in Table 20 show that the expected latency of faults in the memory in AES is significantly higher than in the other considered workloads. This finding coincides with the results in Table 19, which show that AES has a higher probability of latent fault. These results also indicate that the relationship between the average fault propagation latency and the average fault latency in memory is inversely proportional. This analysis consumed 630.2 Kb of memory and took 9.3 seconds to be performed.

In this work, the criticality of a fault is measured at different levels. In the instruction based analysis, the criticality of each fault is evaluated by the lifetime of this fault. These results are then used to better estimate the total criticality of each fault at each vulnerable site throughout the application. This is done by summing all register lifetimes in all the instructions in which that register is active. This total lifetime is divided by the number of execution cycles necessary to run the application. Similar methods for evaluating the application criticality can be found in other works

Table 20: Average Fault Propagation and Fault Latency.

Workload Subset	Avg. Fault Propagation Latency	Avg. Fault Latency in Memory
QSort	802	5362
CRC32	613	8536
AES	459	27081

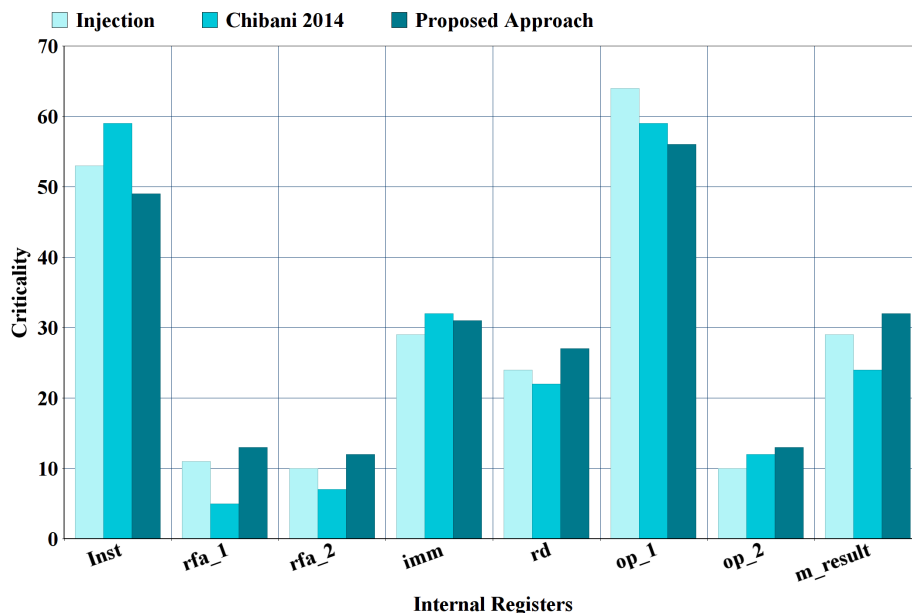


Figure 66: Criticality Evaluation of the LEON3 Pipeline Registers (CRC Benchmark). Injection and Chibani 2014 results reproduced from [40]

such as [40]. An example of this analysis is shown in Fig. 66. This figure depicts the criticality of the internal registers of the integer pipeline considering the CRC application. It can be observed that based on the flow of the application registers have different criticalities. For instance, for this application *op_1* and *imm* registers are more critical than *op_2*. Fig. 66 shows a comparison between fault injection and predictive lifetimes obtained with the technique proposed in [40] for the CRC computation. The fault injection results are based on the experiment done in [40].

As the proposed approach is based on conducting system-level analyses, it is expected to have some margin of over- or under-approximation. Based on the results in Fig. 66, it can be observed that a very good correlation is obtained for all registers

Table 21: Analysis Time for Criticality Evaluation

Benchmark	Fault Injection (minutes)	Chibani et al. (minutes)	Proposed Methodology (Seconds)	Speed-Up (Injection)	Speed-Up (Chibani et al.)
CRC	380	10	37	616	16.2
SHA	690	14	44	941	19
AES	670	15	46	873	19.5
FFT	820	17	52	946	19.6
JPEG	990	18	57	1042	18.9

between our prediction and the fault injection results. Furthermore, Table 21 summarizes the required amount of time to perform the criticality analysis on different applications, compared with the same metric provided by the experiments in [40]. It can be observed that the proposed approach provides speed-ups of up to 1042 times over fault injection and up to 19 times over the technique in [40].

10.2.7 Conclusion

This paper presents a new methodology to assess the reliability of computing systems which takes into consideration both software and the target hardware details. New CTMDP based modeling and analysis of the LEON3 instructions vulnerability to SEUs are proposed. The results of the instruction based analysis are then characterized and then utilized to better estimate the application vulnerability. The proposed modeling and analysis are fully automated using PRISM. Results of the analysis of different types of workloads demonstrate new insights on the fault lifetime and propagation probabilities through the LEON3 are provided at high-level. The proposed methodology allows the designers to evaluate the criticality of different components and variable in the hardware and the application, respectively. It is faster than existing techniques (speed-up of up to 19 times over the technique in [40] and 1042 times over the best previously reported fault injection techniques). This allows iterations without requiring any specific set-up development or dedicated equipment. The accuracy of the proposed methodology is comparable to statistical fault injections. The provided precision is sufficient to identify the most critical flip-flops or execution cycles.

Further works include combining our results with data extracted from radiation testing, in particular, the application cross section (σ_{AP}), to obtain an accurate high-level estimation of the system's sensitivity. Furthermore, based on the results of the proposed analysis, designers could either add redundancy at hardware or software levels.

Bibliography

- [1] Prism website. In <http://www.prismmodelchecker.org>.
- [2] The compass - technical guide to boston scientific cardiac rhythm management products. 2007.
- [3] Atmel (2011). “8-bit avr atmega103 user manual”. Available: <http://www.atmel.com/images/doc0945.pdf>.
- [4] J. A. Abraham and D. P. Siewiorek. An algorithm for the accurate reliability evaluation of triple modular redundancy networks. *IEEE Transactions on Computers.*, 100(7):682–692, 1974.
- [5] P. Adell et al. Analysis of single-event transients in analog circuits. *IEEE Transactions on Nuclear Science*, 47(6):2616–2623, 2000.
- [6] W. Ahmed and O. Hasan. Towards formal fault tree analysis using theorem proving. In *International Conference, CICM*, pages 13–17, 2015.
- [7] R. Alur and T. A. Henzinger. Reactive modules. *Formal Methods in System Design*, 15(1):7–48, 1999.
- [8] Rajeev Alur and David L Dill. A theory of timed automata. *Theoretical computer science*, 126(2):183–235, 1994.
- [9] M. Ammar et al. “Comprehensive vulnerability analysis of systems exposed to seus via probabilistic model checking”. Bremen, Germany, 2016.
- [10] M. Ammar et al. Efficient probabilistic fault tree analysis of safety critical systems via probabilistic model checking. In *Forum on Specification and Design Languages (FDL)*, pages 1–8, 2016.

- [11] M. Ammar et al. System-level analysis of the vulnerability of processors exposed to single-event upsets via probabilistic model checking. *IEEE Transactions on Nuclear Science*, 64(9):2523–2530, 2017.
- [12] D. Ardagna, C. Ghezzi, and R. Mirandola. Rethinking the use of models in software architecture. In *4th International Conference on the Quality of Software-Architectures*, pages 1–27, 2008.
- [13] F. Arnold, A. Belinfante, F. Berg, D. Guck, and M. Stoelinga. Dftcalc: a tool for efficient fault tree analysis (extended version). 2013.
- [14] F. Arnold, A. Belinfante, F. Van der Berg, D. Guck, and M. Stoelinga. Dftcalc: A tool for efficient fault tree analysis. In *32nd International Conference, SAFECOMP*, pages 293–301, 2013.
- [15] J. Asok Kumar. Statistical guarantees of performance for rtl designs. 2012.
- [16] D. Auder et al. Implementing fault injection and tolerance mechanisms in multiprocessor systems. In *Defect and Fault Tolerance in VLSI Systems*, pages 310–317, 1996.
- [17] J. R. Azambuja et al. “*Hybrid Fault Tolerance Techniques to Detect Transient Faults in Embedded Processors*”. Springer, 2014.
- [18] Rodrigo Bagur, Mathilde Chamula, Émilie Brouillard, Caroline Lavoie, Luis Nombela-Franco, Anne-Sophie Julien, Louis Archambault, Nicolas Varfalvy, Valérie Gaudreault, Sébastien X Joncas, et al. Radiotherapy-induced cardiac implantable electronic device dysfunction in patients with cancer. *The American Journal of Cardiology*, 2016.
- [19] C. Baier et al. “*Principles of model checking*”. The MIT Press, 2008.
- [20] C. Baier and J.P. Katoen. Principles of model checking. 2008.
- [21] G. Bany Hamad, M. Ammar, O. Ait Mohamed, and Y. Savaria. New insights into soft-faults induced cardiac pacemakers malfunctions analyzed at system-level via model checking. *IEEE Access*, 6:62107–62119, 2018.

- [22] G. Bany Hamad et al. New insights into the single event transient propagation through static and tspe logic. *IEEE Transactions on Nuclear Science*, 61(4):1618–1627, 2014.
- [23] G. Bany Hamad et al. “Characterizing, modeling, and analyzing soft error propagation in asynchronous and synchronous digital circuits”. *Microelectronics Reliability*, vol. 55, no. 1, pp. 238–250, 2015.
- [24] Gerd Behrmann, Alexandre David, Kim Guldstrand Larsen, John Hakansson, Paul Petterson, Wang Yi, and Martijn Hendriks. Uppaal 4.0. In *Third International Conference on the Quantitative Evaluation of Systems-(QEST’06)*, pages 125–126. IEEE, 2006.
- [25] Roy Beinart and Saman Nazarian. Effects of external electrical and magnetic fields on pacemakers and defibrillators from engineering principles to clinical practice. *Circulation*, 128(25):2799–2809, 2013.
- [26] F. Bezerra et al. “SEU and latchup results on transputers”. *IEEE Trans. Nucl. Sci.*, vol. 43, no. 3, pp. 893–898, 1996.
- [27] Andrea Bianco and Luca De Alfaro. Model checking of probabilistic and non-deterministic systems. In *International Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 499–513. Springer, 1995.
- [28] F. Bidner. Fault tree analysis of the hermes cubesat. *University of Colorado at Boulder, USA*, 2010.
- [29] C. Bottoni et al. Heavy ions test result on a 65nm sparv8 radiation-hard microprocessor. In *IEEE International Reliability Physics Symposium*, pages 5F–5, 2014.
- [30] H. Boudali, P. Crouzen, and M. Stoelinga. Dynamic fault tree analysis using input/output interactive markov chains. In *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 708–717, 2007.
- [31] H. Boudali and J.B. Duga. A new bayesian network approach to solve dynamic fault trees. In *Annual Reliability and Maintainability Symposium*, pages 451–456, 2005.

- [32] M. Bouissou and J.L. Bon. A new formalism that combines advantages of fault-trees and markov models: Boolean logic driven markov processes. *Reliability Engineering & System Safety*, 82(2):149–163, 2003.
- [33] M. Bozzano, A. Cimatti, and C. Mattarei. Automated analysis of reliability architectures. In *18th International Conference on Engineering of Complex Computer Systems (ICECCS)*, pages 198 – 207, 2013.
- [34] M. Bozzano, A. Cimatti, and C. Mattarei. Efficient analysis of reliability architectures via predicate abstraction. In *9th International Haifa Verification Conference (HVC)*, pages 279–294, 2013.
- [35] M. Bozzano, A. Cimatti, and F. Tapparo. Symbolic fault tree analysis for reactive systems. In *5th International Symposium on Automated Technology for Verification and Analysis (ATVA)*, pages 162–176, 2007.
- [36] PD Bradley and E Normand. Single event upsets in implantable cardioverter defibrillators. *IEEE Transactions on Nuclear Science*, 45(6):2929–2940, 1998.
- [37] J. D. Carpinelli. “*Computer systems organization and Architecture*”, 1st ed. Addison-Wesley Longman Publishing Co., MA, USA, 2000.
- [38] Taolue Chen, Marco Diciolla, Marta Kwiatkowska, and Alexandru Mereacre. Quantitative verification of implantable cardiac pacemakers. In *Real-Time Systems Symposium (RTSS), 2012 IEEE 33rd*, pages 263–272. IEEE, 2012.
- [39] E. Cheshmikhani et al. Probabilistic analysis of dynamic and temporal fault trees using accurate stochastic logic gates. *Microelectronics Reliability*, 55(11):2468–2480, 2015.
- [40] K Chibani, M Ben-Jrad, Michele Portolan, and Régis Leveugle. Fast accurate evaluation of register lifetime and criticality in a pipelined microprocessor. In *IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, pages 1–6, 2014.
- [41] Frank Ciesinski and Marcus Größer. On probabilistic computation tree logic. In *Validation of Stochastic Systems*, pages 147–188. Springer, 2004.

- [42] Gari D Clifford, Shamim Nemati, and Reza Sameni. An artificial vector model for generating abnormal electrocardiographic rhythms. *Physiological measurement*, 31(5):595, 2010.
- [43] David Crocker. Perfect developer: a tool for object-oriented formal specification and refinement. *Tools exhibition notes at formal methods Europe*, 2003.
- [44] A. Da Silva et al. Leon3 vip: A virtual platform with fault injection capabilities. In *DSD*, pages 813–816, 2010.
- [45] M. Daněk et al. The leon3 processor. In *UTLEON3: Exploring Fine-Grain Multi-Threading in FPGAs*, pages 9–14. Springer, 2013.
- [46] A. David et al. Uppaal smc tutorial. *International Journal on Software Tools for Technology Transfer*, 17(4):397–415, 2015.
- [47] C. Dehnert et al. A storm is coming: A modern probabilistic model checker. In *International Conference on Computer Aided Verification*, pages 592–600. Springer, 2017.
- [48] Y. S. Dhillon et al. *Soft-Error Tolerance Analysis and Optimization of Nanometer Circuits*, chapter 28, pages 389–400. Springer Netherlands, 2008.
- [49] J. B. Dugan et al. Fault trees and sequence dependencies. In *Reliability and Maintainability Symposium*, pages 286–293, 1990.
- [50] J. B. Dugan et al. Dynamic fault-tree models for fault-tolerant computer systems. *IEEE Transactions on Reliability*, 41(3):363–377, 1992.
- [51] K. Durga Rao, V. Gopika, V.V.S. Sanyasi Rao, H.S. Kushwaha, A.K. Verma, and A. Srividya. Dynamic fault tree analysis using monte carlo simulation in probabilistic safety assessment. In *Reliability Engineering and System Safety Volume 94, Issue 4*, page 872–883, 2009.
- [52] J. Faulin et al. *Simulation methods for reliability and availability of complex systems*. Springer Science & Business Media, 2010.
- [53] J. Faulin Fajardo, A.A. Juan Perez, S.S. Martorell Alsina, and J.E Ramirez-Marquez. Simulation methods for reliability and availability of complex systems. 2010.

- [54] M. Favalli and C. Metra. Tmr voting in the presence of crosstalk faults at the voter inputs. *IEEE Transactions on Reliability.*, 53(3):342–348, 2004.
- [55] P. Ferreyra et al. “Injecting single event upsets in a digital signal processor by means of direct memory access requests”. In *Radiation and its Effects on Devices and Systems*, Grenoble, France, 2001, pp. 248–252.
- [56] P. A. Ferreyra et al. “Failure map functions and accelerated mean time to failure tests: New approaches for improving the reliability estimation in systems exposed to single event upsets”. *IEEE Trans. Nucl. Sci.*, vol. 52, no. 1, pp. 494–500, 2005.
- [57] P. A. Ferreyra et al. “Failure and coverage factors based markoff models: A new approach for improving the dependability estimation in complex fault tolerant systems exposed to seus”. *IEEE Trans. Nucl. Sci.*, vol. 54, no. 4, pp. 912–919, 2007.
- [58] A. M. Ferrick, N. Bernstein, A. Aizer, and L. Chinitz. Cosmic radiation induced software electrical resets in icds during air travel. *Heart Rhythm*, 5(8):1201–1203, 2008.
- [59] V. Forejt, M. Kwiatkowska, G. Norman, and D. Parker. Automated verification techniques for probabilistic systems. In *Formal Methods for Eternal Networked Software Systems*, pages 53–113. Springer, 2011.
- [60] V. Forejt, M. Kwiatkowska, G. Norman, and D. Parker. *Formal Methods for Eternal Networked Software Systems*. 2011.
- [61] S. Friedenthal, A. Moore, and R. Steiner. *A practical guide to SysML: the systems modeling language*. Morgan Kaufmann, 2014.
- [62] J. Gaisler et al. Grlib ip core user’s manual. version 1.3. 7-b4144. *Gaisler research*, <http://gaisler.com/products/grlib/grip.pdf>, 2014.
- [63] Daniel D Gajski, Samar Abdi, Andreas Gerstlauer, and Gunar Schirner. *Embedded system design: modeling, synthesis and verification*. Springer Science & Business Media, 2009.

- [64] C. Geng et al. Monte carlo simulation based on geant4 of single event upset induced by heavy ions. *Mechanics and Astronomy Science China Physics*, 56(6):1120–1125, 2013.
- [65] P. Godefroid et al. “*Partial-Order Methods for the Verification of Concurrent Systems*”, volume 1032 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 1996.
- [66] A. O. Gomes and M. V. M. Oliveira. Formal specification of a cardiac pacing system. In *International Symposium on Formal Methods*, pages 692–707. Springer, 2009.
- [67] J. Grinschgl et al. Modular fault injector for multiple fault dependability and security evaluations. In *Digital System Design (DSD)*, pages 550–557, 2011.
- [68] M. Guthaus et al. Mibench: A free, commercially representative embedded benchmark suite. In *IEEE WWC-4*, pages 3–14, 2001.
- [69] H. Guzman-Miranda et al. Noninvasive fault classification, robustness and recovery time measurement in microprocessor-type architectures subjected to radiation-induced errors. *IEEE Transactions on Instrumentation and Measurement*, 58(5):1514–1524, 2009.
- [70] M. Hamamatsu, T. Tsuchiya, and T. Kikuno. On the reliability of cascaded tmr systems. In *IEEE 16th Pacific Rim International Symposium on Dependable Computing (PRDC)*., pages 184–190, 2010.
- [71] Haruko Hashii, Takayuki Hashimoto, Ayako Okawa, Koichi Shida, Tomonori Isobe, Masahiro Hanmura, Tetsuo Nishimura, Kazutaka Aonuma, Takeji Sakae, and Hideyuki Sakurai. Comparison of the effects of high-energy photon beam irradiation (10 and 18 mv) on 2 types of implantable cardioverter-defibrillators. *International Journal of Radiation Oncology* Biology* Physics*, 85(3):840–845, 2013.
- [72] K. A. Hoque, O. Ait Mohamed, and Y. Savaria. Towards an accurate reliability, availability and maintainability analysis approach for satellite systems based on probabilistic model checking. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, pages 1635–1640.

- [73] Coen W Hurkmans, Egon Scheepers, Bob GF Springorum, and Hans Uiterwaal. Influence of radiotherapy on the latest generation of implantable cardioverter-defibrillators. *International Journal of Radiation Oncology, Biology, Physics*, 63(1):282–289, 2005.
- [74] SPARC International. The sparc architecture manual version 8. *SPARC International Inc*, 1998.
- [75] Z. Jiang, M. Pajic, R. Alur, and R. Mangharam. Closed-loop verification of medical devices with model abstraction and refinement. *International Journal on Software Tools for Technology Transfer*, 16(2):191–213, 2014.
- [76] Z. Jiang, M. Pajic, S. Moarref, R. Alur, and R. Mangharam. Modeling and verification of a dual chamber implantable pacemaker. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 188–203. Springer, 2012.
- [77] Zhihao Jiang, Miroslav Pajic, and Rahul Mangharam. Model-based closed-loop testing of implantable pacemakers. In *Proceedings of the 2011 IEEE/ACM Second International Conference on Cyber-Physical Systems*, pages 131–140. IEEE Computer Society, 2011.
- [78] Zhihao Jiang, Miroslav Pajic, and Rahul Mangharam. Cyber-physical modeling of implantable cardiac medical devices. *Proceedings of the IEEE*, 100(1):122–137, 2012.
- [79] Jonathan M. Johnson and Michael J. Wirthlin. Voter insertion algorithms for fpga designs using triple modular redundancy. In *Proceedings of the 18th annual ACM/SIGDA international symposium on Field programmable gate arrays*, pages 249–258. ACM, 2010.
- [80] A. H. Johnston et al. A model for single-event transients in comparators. *IEEE Transactions on Nuclear Science*, 47(6):2624–2633, 2000.
- [81] S. Kabir et al. Fuzzy temporal fault tree analysis of dynamic systems. *International Journal of Approximate Reasoning*, 77:20–37, 2016.

- [82] F. Kastensmidt and P. Rech. Radiation effects and fault tolerance techniques for fpgas and gpus. In *FPGAs and Parallel Architectures for Aerospace Applications*, pages 3–17. Springer, 2016.
- [83] F. L. Kastensmidt et al. “On the use of vhdl simulation and emulation to derive error rates”. In *Radiation and Its Effects on Components and Systems*, Grenoble, France, 2001, pp. 253–260.
- [84] F. L. Kastensmidt et al. “*Fault-tolerance techniques for SRAM-based FPGAs*”, volume 32. Springer, 2006.
- [85] F. Kerryann et al. “IRT: a modeling system for single event upset analysis that captures charge sharing effects”. In *IEEE International Reliability Physics Symposium*, pages 5F–1, 2014.
- [86] Maha Kooli, Firas Kaddachi, Giorgio Di Natale, and Alberto Bosio. Cache-and register-aware system reliability evaluation based on data lifetime analysis. In *IEEE VLSI Test Symposium (VTS)*, pages 1–6, 2016.
- [87] M. Kwiatkowska et al. Performance analysis of probabilistic timed automata using digital clocks. *Formal Methods in System Design*, 29(1):33–78, 2006.
- [88] M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic symbolic model checking with prism: A hybrid approach. *International Journal on Software Tools for Technology Transfer*, 6(2):128–142, 2004.
- [89] M. Kwiatkowska, G. Norman, and D. Parker. Symmetry reduction for probabilistic model checking. In *International Conference on Computer Aided Verification*, pages 234–248. Springer, 2006.
- [90] M. Kwiatkowska, G. Norman, and D. Parker. Advances and challenges of probabilistic model checking. In *48th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*., pages 1691–1698, 2010.
- [91] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Proc. 23rd International Conference on Computer Aided Verification*, volume 6806, pages 585–591, 2011.

- [92] M. Kwiatkowska, G. Norman, and D. Parker. Prism 4.0: Verification of probabilistic real-time systems. In *Computer aided verification*, pages 585–591. Springer, 2011.
- [93] T. Lanfang, T. Qingping, and L. Jianli. Specification and verification of the triple-modular redundancy fault tolerant system using csp. In *The Fourth International Conference on Dependability*, pages 14–17, 2011.
- [94] M. Learn. Evaluation of the leon3 soft-core processor within a xilinx radiation hardened field programmable gate array. Technical report, Sandia National Laboratories (SNL-NM), Albuquerque, NM (United States), 2011.
- [95] P. A. Lee and T.s Anderson. *Fault tolerance: principles and practice*, volume 3. Springer Science & Business Media, 2012.
- [96] W.S. Lee, D.L. Grosh, F.A. Tillman, and Lie C.H. Fault tree analysis, methods, and applications - a review. pages 194 – 203, 1985.
- [97] T. Mähne and A. Vachoux. Proposal for a bond graph based model of computation in systemc-ams. In *Proceedings of the Tenth International Forum on Specification and Design Languages*, pages 25–31, 2007.
- [98] A. Malinowski et al. Comparison of embedded system design for industrial applications. *IEEE transactions on industrial informatics*, 7(2):244–254, 2011.
- [99] F. Mhenni, N. Nguyen, and J.Y. Choley. Automatic fault tree generation from sysml system models. In *IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*., pages 715–720, 2014.
- [100] T. Mähne, A. Vachoux, and Villar E. Proposal for a bond graph based model of computation in systemc-ams. pages 25–31, 2007.
- [101] N. Miskov-Zivanov et al. Mars-c: modeling and reduction of soft errors in combinational circuits. In *Proceedings of Design Automation Conference (DAC)*, pages 767–772, 2006.
- [102] Harry G Mond and Alessandro Proclemer. The 11th world survey of cardiac pacing and implantable cardioverter-defibrillators: Calendar year 2009–a

- world society of arrhythmia's project. *Pacing and clinical electrophysiology*, 34(8):1013–1027, 2011.
- [103] S. Mukherjee. *Architecture design for soft errors*. Morgan Kaufmann, 2011.
- [104] L. H. Mutuel. Single Event Effects Mitigation Techniques Report. Technical report, Federal Aviation Administration William J. Hughes Technical Center, 2016.
- [105] G. Natale et al. Reliability of computing systems: From flip flops to variables. In *IOLTS*, pages 196–198, 2017.
- [106] J.R. Norris. Markov chains. In *Cambridge University Press*, 1997.
- [107] D. Ortega et al. Runaway pacemaker: A forgotten phenomenon? *EP Europace*, 7(6):592–597, 2005.
- [108] G. K. Palshikar. Temporal fault trees. *Information and Software Technology*, 44(3):137–150, 2002.
- [109] Z. Peng et al. Risk assessment of railway transportation systems using timed fault trees. *Quality and Reliability Engineering International*, 32(1):181–194, 2016.
- [110] F. Piedad and M. Hawkins. *High availability: design, techniques, and processes*. Prentice Hall Professional, 2001.
- [111] S. L. Pinski and R. G. Trohman. Interference in implanted cardiac devices, part ii. *Pacing and clinical electrophysiology*, 25(10):1496–1509, 2002.
- [112] A. Rae et al. A behaviour-based method for fault tree generation. In *Int. System Safety Conference, System Safety Society*, pages 289–298, 2004.
- [113] K. D. Rao, V. Gopika, V.S. Rao, H.S. Kushwaha, A. K. Verma, and A. Srividya. Dynamic fault tree analysis using monte carlo simulation in probabilistic safety assessment. *Reliability Engineering & System Safety*, 94(4):872–883, 2009.
- [114] K.D. Raa, V. Gopikaa, V.V.S. Raa, H.S. Kushwahaa, A.K. Vermab, and A. Srividyaab. Dynamic fault tree analysis using monte carlo simulation in

- probabilistic safety assessment. In *Reliability Engineering and System Safety, Volume 94, Issue 4*, page 872–883, 2009.
- [115] Amy G Rapsang and Prithwis Bhattacharyya. Pacemakers and implantable cardioverter defibrillators-general and anesthetic considerations. *Revista Brasileira de Anestesiologia*, 64(3):205–214, 2014.
- [116] R. Reis et al. “*Circuit Design for Reliability*”. Springer New York, NY, 2015.
- [117] S. Rezgui et al. “Estimating error rates in processor-based architectures”. *IEEE Trans. Nucl. Sci.*, vol. 48, no. 5, pp. 1680–1687, 2001.
- [118] A. Rohani et al. “A technique for accelerating injection of transient faults in complex socs”. In *14th Euromicro Conference on Digital System Design*, Oulu, Finland, Aug, 2011, pp. 213–220.
- [119] E. Ruijters et al. Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools. *Computer science review*, 15:29–62, 2015.
- [120] S. J. Schilling. Contribution to temporal fault tree analysis without modularization and transformation into the state space. *arXiv preprint arXiv:1505.04511*, 2015.
- [121] D.H. Shin, S. He, and J. Zhang. Robust and cost-effective design of cyber-physical systems: An optimal middleware deployment approach. *IEEE/ACM Transactions on Networking*, 24(2):1081–1094, 2015.
- [122] Y. Shoukry et al. Secure state estimation for cyber-physical systems under sensor attacks: A satisfiability modulo theory approach. *IEEE Transactions on Automatic Control*, 62(10):4917–4932, 2017.
- [123] Sana Shuja, Sudarshan K Srinivasan, Shaista Jabeen, and Dharmakeerthi Nawarathna. A formal verification methodology for ddd mode pacemaker control programs. *Journal of Electrical and Computer Engineering*, 2015:57, 2015.
- [124] R.M. Sinnamon and J.D. Andrews. Improved accuracy in quantitative fault tree analysis. In *Proceedings of the 12th Advances in Reliability Technology Symposium, Manchester, UK*, 1996.

- [125] R.M. Sinnamon and J.D. Andrews. Improved efficiency in qualitative fault tree analysis. In *Proceedings of the 12th Advances in Reliability Technology Symposium, Manchester, UK*, 1996.
- [126] SK Souliman and J Christie. Pacemaker failure induced by radiotherapy. *Pacing and Clinical Electrophysiology*, 17(3):270–273, 1994.
- [127] F. Sturesson et al. Radiation characterization of a dual core leon3-ft processor. In *European Conference on Radiation and Its Effects on Components and Systems, 2011*, pages 938–944.
- [128] K. J. Sullivan, J. B. Dugan, and D. Coppit. The galileo fault tree analysis tool. In *IEEE Twenty-Ninth Annual International Symposium on Fault-Tolerant Computing.*, pages 232–235, 1999.
- [129] D. D. Thaker, R. Amirtharajah, F. Impens, I.L. Chuang, and F. T. Chong. Recursive tmr: Scaling fault tolerance in the nanoscale era. *IEEE Design & Test of Computers.*, 22(4):298–305, 2005.
- [130] Alexandre Trigano, Guillaume Hubert, Jannie Marfaing, and Karine Castellani. Experimental study of neutron-induced soft errors in modern cardiac pacemakers. *Journal of interventional cardiac electrophysiology*, 33(1):19–25, 2012.
- [131] L. A. Tuan, M. C. Zheng, and Q. T. Tho. Modeling and verification of safety critical systems: A case study on pacemaker. In *Fourth International Conference on Secure Software Integration and Reliability Improvement*, pages 23–32, 2010.
- [132] A. Valmari. The state explosion problem. In *Lectures on Petri nets I: Basic models*, pages 429–528. Springer, 1998.
- [133] R. Velazco et al. Heavy ion test results for the 68020 microprocessor and the 68882 coprocessor. *IEEE TNS*, 1992.
- [134] R. Velazco et al. “Heavy ion test results for the 68020 microprocessor and the 68882 coprocessor”. *IEEE Trans. Nucl. Sci.*, vol. 39, no. 3, pp. 436–440, 1992.

- [135] R. Velazco et al. “Predicting error rate for microprocessor based digital architectures through C.E.U. (code emulating upsets) injection”. *IEEE Trans. Nucl. Sci.*, vol. 47, no. 6, pp. 2405–2411, 2000.
- [136] R. Velazco, D. McMorrow, and J. Estela. *Radiation Effects on Integrated Circuits and Systems for Space Applications*. Springer International Publishing, 2019.
- [137] W. E. Vesely et al. Fault tree handbook. Technical report, Nuclear Regulatory Commission Washington dc, 1981.
- [138] W.E. Vesely, F.F. Goldberg, N.H. Roberts, and D.F. Haasl. Fault tree handbook (nureg-0492). In <http://www.hq.nasa.gov/office/codeq/doctree/fthb.pdf>, 1981.
- [139] M. Volk, S. Junges, and J.P. Katoen. Advancing dynamic fault tree analysis. *arXiv preprint arXiv:1604.07474*, 2016.
- [140] Robert M Wachter, Lee Goldman, and Harry Hollander. *Hospital medicine*. Lippincott Williams & Wilkins, 2005.
- [141] A. Wald. Sequential tests of statistical hypotheses. *The annals of mathematical statistics*, 16(2):117–186, 1945.
- [142] M. Walker et al. Compositional temporal fault tree analysis. *Computer safety, reliability, and security*, pages 106–119, 2007.
- [143] M. Walker et al. Synthesis and analysis of temporal fault trees with pandora: The time of priority and gates. *Nonlinear Analysis: Hybrid Systems*, 2(2):368–382, 2008.
- [144] M. Walker et al. A hierarchical method for the reduction of temporal expressions in pandora. In *Proceedings of the First Workshop on Dynamic Aspects in Dependability Models for Fault-Tolerant Systems*, pages 7–12, 2010.
- [145] M. Webster, M. Fisher, N. Cameron, and M. Jump. Towards certification of autonomous unmanned aircraft using formal model checking and simulation. In *30th International Conference, SAFECOMP*, 2011.

- [146] P. G. Wijayarathna et al. Extending fault trees with an and-then gate. In *International Symposium on Software Reliability Engineering*, pages 283–292, 2000.
- [147] J. Wu, S. Yan, and L. Xie. Reliability analysis method of a solar array by using fault tree analysis and fuzzy reasoning petri net. In *Acta Astronautica Volume 69, Issues 11–12*, page 960–968, 2011.
- [148] T. Zaremba, A. R. Jakobsen, M. Søgaaard, A. M. Thøgersen, and S. Riahi. Radiotherapy in patients with pacemakers and implantable cardioverter defibrillators: a literature review. *Europace Journal*, pages 135–145, 2015.
- [149] Tomas Zaremba, Annette Ross Jakobsen, Mette Søgaaard, ANNA THØGERSEN, Martin Berg Johansen, Lærke Bruun Madsen, and Sam Riahi. Risk of device malfunction in cancer patients with implantable cardiac device undergoing radiotherapy: A population-based cohort study. *Pacing and Clinical Electrophysiology*, 38(3):343–356, 2015.
- [150] Tomas Zaremba, Annette Ross Jakobsen, Mette Søgaaard, Anna Margrethe Thøgersen, and Sam Riahi. Radiotherapy in patients with pacemakers and implantable cardioverter defibrillators: a literature review. *Europace*, 18(4):479–491, 2016.
- [151] Tomas Zaremba, Annette Ross Jakobsen, Anna Margrethe Thøgersen, Lars Oddershede, and Sam Riahi. The effect of radiotherapy beam energy on modern cardiac devices: an in vitro study. *Europace*, 16(4):612–616, 2014.
- [152] M. Zhang, Z. Liu, C. Morisset, and A. P. Ravn. Design and verification of fault-tolerant components. In *Methods, Models and Tools for Fault Tolerance*, pages 57–84. 2009.