Thesis for The Degree of Doctor of Philosophy

Towards Accurate Estimation of Error Sensitivity in Computer Systems

Fatemeh Ayatolahi

Department of Computer Science & Engineering
Chalmers University of Technology and Gothenburg University
Gothenburg, Sweden, 2021

Errata

Towards Accurate Estimation of Error Sensitivity in Computer Systems

Fatemeh Ayatolahi
Department of Computer Science & Engineering
Chalmers University of Technology
Gothenburg, Sweden

This errata sheet lists the corrections for the doctoral thesis written by Fatemeh Ayatolahi, titled *Towards Accurate Estimation of Error Sensitivity in Computer Systems*, ISBN 978-91-7905-493-9.

	Location	Original text	Correction
1	Page ii		Add: Technical report number 203D
2	Chapter 1, page 7	eld	field
3	Chapter 1, page 9	dierent	different
4	Chapter 1, page 15	how defers to use	The authors compared the use
5	Chapter 1, page 15	or satisfying the margin of error.	For satisfying a given margin of error.
6	Paper E, page 4	are meaningful	Are only meaningful
7	Paper E, page 4	implements	implement
8	Paper E, page 7	hypothesis	hypothesis testing
9	Paper E, page 10	section ??	section 5.3
10	Paper E, page 12	On the other hand, BitCnt2 resulted in significantly different error sensitivity according to statistics.	On the other hand, the statistical test for BitCnt2 suggests to *reject* the null hypothesis.
11	Paper E, page 14	We	we
12	Paper E, page 14	reveal	result in
13	Paper E, page 16	approximately	an approximately
14	Chapter 2, first blue page	PAPER I	PAPER A

Towards Accurate Estimation of Error Sensitivity in Computer Systems

Fatemeh Ayatolahi

ISBN 978-91-7905-493-9
Doktorsavhandlingar vid Chalmers tekniska högskola, Ny serie nr 4960.
ISSN 0346-718X
Department of Computer Science & Engineering
Chalmers University of Technology and Gothenburg University
Gothenburg, Sweden

This thesis has been prepared using LaTeX.
Printed by Chalmers Reproservice,
Gothenburg, Sweden 2021.

"To those who, in spite of everything, still choose goodness"
- Marie Lu, The Midnight Star

Towards Accurate Estimation of Error Sensitivity in Computer Systems

Fatemeh Ayatolahi
Department of Computer Science and Engineering
Chalmers University of Technology, Sweden

Abstract
Fault injection is an increasingly important method for assessing, measuring and observing the system-level impact of hardware and software faults in computer systems. This thesis presents the results of a series of experimental studies in which fault injection was used to investigate the impact of bit-flip errors on program execution. The studies were motivated by the fact that transient hardware faults in microprocessors can cause bit-flip errors that can propagate to the microprocessors instruction set architecture registers and main memory. As the rate of such hardware faults is expected to increase with technology scaling, there is a need to better understand how these errors (known as 'soft errors') influence program execution, especially in safety-critical systems.

Using ISA-level fault injection, we investigate how five aspects, or factors, influence the error sensitivity of a program. We define *error sensitivity* as the conditional probability that a bit-flip error in live data in an ISA-register or main-memory word will cause a program to produce silent data corruption (SDC; i.e., an erroneous result). We also consider the estimation of a measure called *SDC count*, which represents the number of ISA-level bit flips that cause an SDC.

The five factors addressed are (a) the inputs processed by a program, (b) the level of compiler optimization, (c) the implementation of the program in the source code, (d) the fault model (single bit flips vs double bit flips) and (e) the fault-injection technique (inject-on-write vs inject-on-read). Our results show that these factors affect the error sensitivity in many ways; some factors strongly impact the error sensitivity or SDC count whereas others show a weaker impact. For example, our experiments show that single bit flips tend to cause SDCs more than double bit flips; compiler optimization positively impacts the SDC count but not necessarily the error sensitivity; the error sensitivity varies between 20% and 50% among the programs we tested; and variations in input affect the error sensitivity significantly for most of the tested programs.

Keywords
soft errors, fault injection, error sensitivity, silent data corruption

Acknowledgment

It is a great pleasure to express gratitude to all people supported and helped me during my good and hard days in my studies and in my life.
I would like to express my deepest gratitude to my supervisor, Johan Karlsson, for inspiring me to continue my studies in functional safety field. Thank you for giving me this opportunity and encouraging me to pursue my research to get my PhD. Thank you for all your supports and invaluable guidance.
Special thanks are due to Behrooz Sangchoolie, with whom I have the pleasure to start this journey with joint work for master thesis and then we become a great team to collaborate, to motivate each other through the way and exchange insights.
Also special thanks to Domenico Di Leo for his great support and collaboration in my master thesis which inspired me to continue my studies in fault injection field. Indeed, special thanks to the countless support from Daniel Skarin and Roger Johansson to know more about Goofi-2 and troubleshoot hardware and compiler problems. I would also like to thank Raul Barbosa and Jonny Vinetr for their valuable discussions about fault injection.
I would like take this opportunity to thank BeSafe project team Mafijul Md. Islam1, Daniel Skarin, Jonny Vinter, Fredrik Törner, Andreas Käck, Mattias Nyberg, Johan Haraldsson, Patrik Isaksson, Mats Olsson, for interesting discussions, valuable feedback, and joyful meetings on benchmarking of functional safety in the automotive industry and ISO26262.
Many thanks to my friends and colleagues in department, administrative supports in the department, professors and students. I would like to mention all names but I'm afraid to miss some, my colleagues at 4th floor (computer engineering division), PhD Council, PhD fika, persian fika, TA teams,... you all made this department a great place to work and have fun!
Life is too short to be anything but happy, to be anything but you!
Special thanks definitely goes to friends and family. My dear friends from childhood, school, university, master studies, summer schools, conferences, who are now all around the world! thanks for being so kind, caring, encouraging and understanding.
My dear wonderful mom, I am blessed to have been able to look up to you, as a strong, independent, diligent woman. Thanks for constant inspiring and being such an amazing hero in my life. My dear wonderful dad, thanks for all your efforts, caring and being a constant support in every occasion in my life. My lovely sister and brother, my cute nieces and nephews, thanks for all joy, fun and support.

Dearest Sadegh, thanks for your love, your kindness and your amazing support, not only in studies, but always... My lovely Shayan, you made me a mom and I think that's all! you boosted the meaning of life for me, and you give me joy constantly. My lovely Daniel, you are pure joy! Everything seemed very sad because of pandemic and you amazingly changed it for us! Thanks for being so unbelievably helpful by coming to our life! You adorable and charming guys make me find the joy in curiosity again and realize how life is more and more beautiful ever day...

Fatemeh Ayatolahi
Göteborg, June 2021

List of Publications

Appended publications

This thesis is based on the following publications:

[A] Domenico Di Leo, Fatemeh Ayatolahi, Behrooz Sangchoolie, Johan Karlsson, Roger Johansson "On the Impact of Hardware Faults — An Investigation of the Relationship between Workload Inputs and Failure Mode Distributions"
in Proceedings of the 31^{st} *International Conference on Computer Safety, Reliability, and Security (SAFECOMP), Magdeburg, Germany, 2012.*

[B] Behrooz Sangchoolie, Fatemeh Ayatolahi, Raul Barbosa, Roger Johansson, Johan Karlsson "A Study of the Impact of Bit-flip Errors on Programs Compiled with Different Optimization Levels"
in Proceedings of the 10^{th} *European Dependable Computing Conference (EDCC), Newcastle upon Tyne, UK, 2014.*

[C] Fatemeh Ayatolahi, Behrooz Sangchoolie, Roger Johansson, Johan Karlsson "A Study of the Impact of Single Bit-Flip and Double Bit-Flip Errors on Program Execution"
in Proceedings of the 32^{nd} *International Conference on Computer Safety, Reliability, and Security (SAFECOMP), Toulouse, France, 2013.*

[D] Behrooz Sangchoolie, Fatemeh Ayatolahi, Raul Barbosa, Roger Johansson, Johan Karlsson "A Comparison of Inject-on-Read and Inject-on-Write in ISA-Level Fault Injection"
in Proceedings of the 11^{th} *European Dependable Computing Conference (EDCC), Paris, France, 2015.*

[E] Fatemeh Ayatolahi, Johan Karlsson "Statistical Analysis of Fault-Injection Data — A Case Study using Hypothesis Testing"
Technical report, 2021.

Other publications

The following publications were published during my PhD studies. However, they are not appended to this thesis, due to contents overlapping that of appended publications or contents not related to the thesis.

[a] Peter Folkesson, Fatemeh Ayatolahi, Behrooz Sangchoolie, Jonny Vinter, Mafijul Islam, Johan Karlsson "Back-to-Back Fault Injection Testing in Model-Based Development"
in Proceedings of the 34^{th} *International Conference on Computer Safety, Reliability, and Security (SAFECOMP), Toulouse, France, 2015.*

[b] Mafijul Md. Islam, Behrooz Sangchoolie, Fatemeh Ayatolahi, Daniel Skarin, Jonny Vinter, Fredrik Trner, Andreas Kck, Mattias Nyberg, Emilia Villani, Johan Haraldsson, Patrik Isaksson, Johan Karlsson "Towards Benchmarking of Functional Safety in the Automotive Industry"
in the 14^{th} *European Workshop on Dependable Computing (EWDC), Coimbra, Portugal, 2013.*

[c] Behrooz Sangchoolie, Fatemeh Ayatolahi, Raul Barbosa, Roger Johansson and Johan Karlsson "Benchmarking the Hardware Error Sensitivity of Machine Instructions"
in the 9^{th} *IEEEWorkshop on Silicon Errors in Logic - System Effects (SELSE), Stanford, USA, 2013.*

[d] Behrooz Sangchoolie, Fatemeh Ayatolahi and Johan Karlsson "An Investigation of the Fault Sensitivity of Four Benchmark Workloads"
in the 1^{st} *Workshop on Software-Based Methods for Robust Embedded Systems (SOBRES), Braunschweig, Germany, 2012.*

Contents

Chapter 1

Thesis Summary

1.1 Introduction

Computer systems are used in a wide range of applications in which dependability is a major concern. Self-driving cars, airplane auto-pilots, medical devices, systems for railway signalling, power distribution and financial transactions are a few examples of applications where computer failures could have serious consequences.

Modern computer systems are complex artefacts that rely on billions of transistors and millions of lines of code to operate correctly. Due to this complexity, designing and validating dependable computer systems is a major technical challenge.

A key challenge in developing computer systems for critical applications is to define and obtain measures that can accurately describe the degree to which a system is dependable. According to the widely cited taxonomy and terminology presented in [1], *dependability* is a generic concept that spans a variety of related attributes, such as *availability*, *reliability*, *safety*, *integrity* and *maintainability*.

Thus, there is no single measure of dependability; instead, dependability is described in terms of one or more of these attributes, or by other, more specialised attributes, the choice of which depends on the application. For example, safety may not be a relevant design objective in a business-oriented system, in which availability and data integrity are.

Dependability attributes are often expressed in terms of probabilities (or probability distributions) rather than deterministic values, as the occurrence, activation and propagation of faults is primarily random.

To derive numerical values for a dependability attribute, then, computer designers rely on various dependability modelling techniques, such as reliability block diagrams, Markov chain models and Stochastic Petri-net models. These models, in turn, rely on various input parameters that are often derived by analysing field failure data, or data from fault-injection experiments. Examples of such input parameters, which can themselves be regarded as (secondary) dependability attributes, include *failure rates*, *repair rates*, *coverage factors* and *error-sensitivity factors*.

This thesis addresses the use of fault-injection experiments to determine the

sensitivity of programs to bit-flip errors caused by transient hardware faults. Modern integrated circuits are susceptible to bit-flip errors caused by ionising particles, including high-energy neutrons and alpha particles [2, 3]. Such errors are commonly referred to as 'soft errors', as the ionising particles merely alter the data stored in the affected circuit, without causing permanent damage.

Although the potential threats of soft errors are usually ignored in non-critical applications, they are indeed an important potential source of system failures that should be carefully considered by designers of safety-critical and business-critical applications.

However, measuring soft-error sensitivity is a complex task. Although techniques for such measurements have been a subject of academic research since the 1980s, there are currently no standards for determining soft-error sensitivity; most research has focused on evaluating the effectiveness of various error-detection and fault-tolerance mechanisms, or on studying the failure modes that a program exhibits in the presence of soft errors.

Measurements of soft-error sensitivity, as we discuss them in this thesis, are built on the assumption that the program is an executable program (machine code) for a given microprocessor equipped with a given set of memory and I/O circuits. Thus, we consider measurements of soft-error sensitivity to be valid only for a given hardware configuration.

A major challenge in measuring soft-error sensitivity by fault-injection experiments is that the results are sensitive to variations in the experimental setup related to the fault model and the workload input profile (the inputs processed by the program during the experiments), as well as aspects of implementing the program itself, such as the choice of programming language and level of compiler optimizations.

This thesis presents the results of five studies that investigated how factors related to program implementation and the setup of fault-injection experiments affect estimations of soft-error sensitivity. We classify these factors into three types:

Type I Related to the operation of the system, such as:

- Input profile of the system
- State of unused parts of the main memory

Type II Related to the design and implementation of the system, such as:

- Compiler optimizations
- Choice of algorithm for solving a problem
- Choice of source-code implementation, including programming language and programming style

Type III Related to the assumption of the fault-injection experiments

- Fault model
- Time of injection

Specifically, our work addresses the following factors that can affect the estimation of error sensitivity (the type of factor is stated in parentheses):

- Workload input profile (Type I)
- Source-code implementation (Type II)
- Choice of compiler optimization (Type II)
- Fault model: single bit flips vs double bit flips (Type III)
- Time of injection: inject-on-read vs inject-on-write (Type III)

The results of these investigations have been published in four conference papers. In addition, this thesis includes a technical report that provides guidelines on how to employ hypothesis testing in analysing fault-injection data; the report discusses how hypothesis testing can be used to determine whether a certain factor has a statistically significant impact on error sensitivity.

1.1.1 Contributions

The contributions of this thesis are based on four conference papers and one technical report, summarised in Section 1.6. The principle contributions are as follows:

C1: A series of fault-injection experiments to study five factors that can influence the error sensitivity of a program. These factors are (i) inputs processed by a program, (ii) level of compiler optimization, (iii) source-code implementation, (iv) fault model (single bit flips vs double bit flips) and (v) fault-injection technique (inject-on-write vs inject-on-read).

C2: A classification that divides the five factors into three main types, related to (i) the operation of the system, (ii) the design and implementation of the system and (iii) the assumptions of the fault-injection experiments.

C3: A set of guidelines for using statistical inference techniques – specifically, hypothesis testing, to draw conclusions from fault injection data.

C4: An analysis to reduce the fault space of fault-injection experiments.

1.1.2 Thesis Structure

This thesis consists of the present summary, four conference papers and one technical report. In the next section of this summary, Section 1.2, we provide an overview of common dependability measures as well as an overview of experimental and analytical techniques for assessing and ensuring the dependability of computer systems. Section 1.3 discusses how uncertainty and controllable factors affect estimates of error sensitivity and other dependability measures. In Section 1.4, we define how we measure silent data corruptions due to bit-flip errors and how we define our error-sensitivity measure. Section 1.5 discusses the use of statistical inference techniques for drawing conclusions from fault-injection experiments. Section 1.6 summarises the contributions of the papers and the report and provides statements of the contributions made to each paper by the author of this thesis. Finally, conclusions and directions for future research are given in Section 1.7.

1.2 Dependability Measures and Assessment Methods

According to the taxonomy presented in [1], *dependability* is an umbrella concept that encompasses several dependability attributes, such as reliability, availability, safety, integrity and maintainability. Different analytical and experimental methods, techniques and measures have been introduced over the past decades to answer the difficult question of how to assess the dependability of computer systems.

In this section, we provide an overview of common dependability measures and analytical and experimental methods for assessing the dependability of computer systems.

1.2.1 Dependability measures

The authors of [1] provide the following generic definitions of the five dependability attributes:

- Availability: readiness for correct service.
- Reliability: continuity of correct service.
- Safety: absence of catastrophic consequences on the user(s) and the environment.
- Integrity: absence of improper system alterations.
- Maintainability: ability to undergo modifications and repairs.

Due to the randomness associated with faults and failures, it is inappropriate to express the degree to which a system possesses any of these attributes in terms of an absolute or deterministic value; instead, the fulfilment of the attributes should be described using a relative or probabilistic value [1]. To this end, it is common practise to define four of the attributes more precisely in terms of probabilities:

> *Availability* is the probability that a system delivers a correct service at any given time *t*.
>
> *Reliability* is the probability that a system delivers a correct service (without outages or interruptions) during a given period.
>
> *Safety* is the probability that a system is not in a state that threatens humans or the environment.
>
> *Maintainability* is the probability that a system is operational at a given time *t*, given that the system was faulty at time *t0*.

Integrity is slightly different from the other attributes in that it often relates to complex fault situations, including authorised or non-authorised actions by humans, that can lead to improper system alterations. Therefore, there are no widely used, precise probabilistic definitions associated with *integrity*.

Table 1.1: Summary of dependability measures

Measure	Notation	Description	Relation to other measures
Probability density function (*pdf*) of failures	f(t)	*pdf* of a random variable X, where X is the uptime or lifetime for a system	$f(t) = \frac{d}{dt}F(t)$
Failure probability	F(t)	Cumulative distribution function (CDF) of a random variable X, where X is the uptime or lifetime for a system	$F(t) = Prob(X < t)$ $F(t) = \int_0^t f(x)dx$ $F(t) = 1 - R(t)$
Reliability	R(t)	Probability that a system is operational at time t given that the system was operational at $t = 0$.	$R(t) = Prob(X > t)$
Availability	A(t)	Probability that a system is operational at time t.	$\lim_{t\to\infty} A(t) = \frac{MTTF}{MTTF+MTTR} = \frac{MTTF}{MTBF}$
Probability density function (*pdf*) of restoration times	g(t)	*pdf* of a random variable X, where X is the restoration time for a system	$g(t) = \frac{d}{dt}M(t)$
Maintainability	M(t)	Probability that a repairable system is operational at time t given it was non-operational (broken) at time t=0.	$M(t) = \int_0^t g(x)dx$
Failure rate	$\lambda(t)$	Instantaneous failure rate or hazard rate	$\lambda(t) = \frac{f(t)}{R(t)} = \frac{f(t)}{(1-F(t))}$
Repair rate	$\mu(t)$	Instantaneous repair rate	$\mu(t) = \frac{g(t)}{(1-M(t))}$
Mean Time To Failure	MTTF	The expected uptime or lifetime of a system	$MTTF = \int_0^t R(x)dx$
Mean Time To Repair	MTTR	The expected repair time for a broken system	$MTTR = \int_0^t (1 - M(x))dx$
Mean Time Between Failures	MTBF	The expected time between system failures for a repairable system	$MTBF = MTTF + MTTR$
Fault coverage	c	Conditional probability that a system maintains a correct service given that a fault has occurred.	$c = P(System\ recovers \mid Fault\ occurs)$
Error sensitivity	es	The probability that a soft error (transient hardware fault) results in a Silent Data Corruption (SDC).	$es = P(SDC \mid soft\ error\ occurs)$

Of course, this does not prevent system designers from defining and using probabilistic measures of integrity that are tailored to certain systems or use cases; the probability of loss of data in a fault-tolerant database system is an example of a specific measure of (data) integrity.

In addition to the five attributes, it is common to express dependability in terms of expected values of significant events; examples include mean time to failure (MTTF), mean time to repair (MTTR) and mean time between failures (MTBF). The mathematical relations between these expected values and availability, reliability, safety and maintainability are shown in Table 1.1. $\lim_{t\to\infty} A(t)$ is commonly used as a measure of availability. The table also shows the relationship between reliability and the failure-rate function; note that the failure rate assumes a constant value, denoted as λ, when the life time for a system or a component is exponentially distributed. For other life time distributions, the failure rate is a function of time. In addition, the table also includes definitions of several other commonly used measures of secondary dependability attributes, including fault coverage and error sensitivity.

1.2.2 Analytical methods

Analytical methods used to assess and ensure the dependability of computer systems can be categorised into two main groups: i) methods that estimate probabilistic measures of dependability, and ii) methods that determine if a system, component or algorithm is correctly designed or implemented with respect to a formally expressed requirement.

The first category includes techniques such as reliability block diagrams [4], fault trees [5], Markov chains [6], Stochastic Petri nets [7], and Stochastic activity networks [8]. Reliability block diagrams and fault trees are basic methods for estimating the reliability, safety and failure probability of systems featuring redundant components and subsystems. And although they are useful for modelling a variety of redundant systems, RBD's and fault trees cannot be used to model systems that employ dynamic reconfigurations in response to component and subsystem failures.

Markov chains are powerful tools for modelling a wide variety of systems, including those that employ static and dynamic redundancy. One limitation of Markov chains, however, is that they suffer from the state explosion problem, which makes it computationally demanding to solve a Markov chain model of a system that contains many components. However, it is often possible to circumvent the state explosion problem by dividing a complex system into smaller subsystems.

Compared to Markov chains, Stochastic Petri nets provide a more compact way for expressing reliability, availability and safety models. Petri nets are dynamic, which makes them helpful in analysing the dependability of large-scaled and dynamic systems. Stochastic activity networks (SANs) are also a generalised version of Stochastic Petri nets which allows for representation of the concurrency, timelines, degraded functionality, repair ability and dynamics of the system.

It is also possible to conduct descriptive analysis using fault trees. Hazard analysis and risk assessment (HARA), failure mode and effects analysis (FMEA), and fault propagation and transformation calculus (FPTC) [9] are

other examples of techniques for descriptive analysis. These methods may include elements of probabilistic analysis, but are often used to conduct purely qualitative analysis of the behaviour of a system in the presence of component failures. The authors of [10, 11] introduced methods to automate the process of generating fault trees. As manual analysis can be time-consuming and error-prone for complex systems, these methods can effectively improve the quality and reduce the cost of fault tree analyses. Further, systematically reusing an artifact is discussed in [12] which is helpful in conducting safety cases for safety analysis assurance in complex systems.

The second category of methods are *formal methods*. These rely on mathematical formalisms to construct proofs of reasoning to precisely decide whether an algorithm, program or hardware circuit satisfies a set of formally specified requirements. Here, the result of the analysis is usually a yes/no answer rather than a probabilistic measure of some dependability-related property, but there are also variants of formal methods that enable probabilistic analysis of system properties. Formal methods span a variety of mathematical theories, languages and tools. Two important classes of formal techniques are model checking and deductive verification.

Model checking uses finite state machines to concurrently describe and verify a system and its temporal behaviour. The main challenge in this method is dealing with state space explosion in defining complex sequential systems. Probabilistic model checking extends model checking by automatically verifying a system that shows probabilistic (stochastic) behaviour thus giving a quantitative measurement of the system verification [13]. A well-known tool for performing probabilistic model checking is PRISM [14].

Deductive verification is another approach to formal verification. It applies classical specification and proof of higher order logic formalisation using interactive theorem provers such as HOL. These provers can be used in dependability analysis by direct probabilistic principles; therefore, no dependability modelling, such as RBD or fault tree is needed [15]. In other approaches, RBD and fault trees are formalised in higher order logic to perform formal failure analysis [16].

1.2.3 Field failure data analysis

Another important method for investigating the dependability of computer systems is to collect and analyse eld failure data [17, 18]. This type of studies can provide valuable insights about the rate and types of failures that occur during the operation of a computer system. In addition to identifying reliability bottlenecks, analysis of field failure data can also be used to obtain estimates of different dependability attributes, and for validating dependability models.

The results of a large-scale study of field failures in high-performance computer systems is presented in [18]. In this study, the authors present statistics on data such as root cause of failures, the mean time between failures, and the mean time to repair. One interesting observation, is that the average failure rates differ wildly across different systems, ranging from 20-1000 failures per year.

Field failure data analysis is helpful in finding representative fault models in that the representativeness of a simulation or prototype can be evaluated by such data. For example, [19,20] used field failure analysis to find a representative

fault model to emulate software faults, using fault injection experiments.

1.2.4 Fault injection

Fault injection refers to inserting artificial faults in a computer system. Fault injection is commonly used to test or evaluate a system's ability to detect, mask and mitigate faults; to determine the rate and severity of the failure modes a system may exhibit; and to compare the effectiveness of different approaches to implement fault-tolerant systems.

Fault injection tools are divided into three main categories: (i) tools that inject physical faults, (ii) tools that use software or debugging and testing features in microprocessors to inject faults, and (iii) tools for simulation-based fault injection.

The first category includes tools that use pin-level fault injection [21, 22], heavy-ion radiation [23–25], electromagnetic interference [26, 27] and laser beams [28, 29]. These techniques, which were mainly developed and used in the 1990s, can emulate specific types of hardware faults with high accuracy, such as stuck-at-faults affecting the pins of integrated circuits (pin-level injection) and particle-induced soft errors (laser beams and heavy-ion radiation). However, most of these techniques are too costly or otherwise infeasible to use with modern, highly integrated and compact hardware systems.

Another way to inject physical faults into integrated circuits is to expose circuits to neutron beams. This technique is commonly used by circuit manufacturers as well as independent researchers to estimate the soft error rate (SER) of integrated circuits. However, these types of studies are usually not referred to as fault injection experiments rather as neutron beam experiments [30].

The second category includes tools that rely on software or debugging and testing features in microprocessors to inject faults. This type of tools has been widely used in research projects over the last 25 years to assess the impact of both hardware and software faults, and to evaluate the effectiveness of fault tolerance mechanisms. This category of tools can be divided into three subcategories: software-implemented fault injection (SWIFI) [31–36], test port-based fault injection [37–40], and scan chain-implemented fault injection [41, 42].

SWIFI tools use program code executed by the target system to insert errors (state changes) in main memory locations and instruction set architecture (ISA) registers. SWIFI is a versatile technique, as it can be used to mimic the effects of many fault types, including transient and permanent hardware faults [33, 34, 36], software faults [43–46] and security attacks [47].

Test port–based fault injection is similar to SWIFI in that it imitates the effects of faults by inserting errors in main memory locations and ISA-registers. However, unlike SWIFI, it injects faults without altering the software executed by the target system; instead, it injects faults using an external debugger connected to a test port, or a debug port, on the target system's microprocessor. By controlling the debugger with remote commands, the fault injection tool can issue commands to the microprocessor to stop and start program execution, set breakpoints and alter the contents of main memory words and ISA registers. The GOOFI-2 tool, which we used in our experiments, implements test port–based fault injection on a Freescale MPC565 controller, using the NEXUS debugger [48] and winIDEA integrated development environment from

iSYSTEM [49].

Scan-chain implemented fault injection works similarly to test port-based fault injection; it injects faults via a test access port (TAP), which provides a serial interface that makes it possible to write and read the data in any internal register in the circuit included in the scan-chain. Modern microprocessors are equipped with scan-chains and TAPs mainly for pre-shipment chip testing by manufacturers. Since the users of a microprocessor usually do not have access to information about how to operate the TAP, scan-chain—based fault injection has been used in only a few research studies [41, 42].

The third category includes tools that employ simulation models of computers at dierent levels of abstraction [50, 51], ranging from the transistor and micro-architectural levels [51, 52] to high-level functional models of software or hardware modules [53, 54] and even the system level [55]. For example, simulation-based fault injection has been widely used to evaluate the vulnerability of hardware architectures or software codes with respect to radiation-induced soft errors [56, 57]. To speed up fault injection experiments conducted at the micro-architectural level, researchers have explored the possibility of injecting faults into hardware models implemented in field-programmable gate arrays (FPGAs) [58–63]. This technique is referred to as 'hardware emulation–based fault injection'; compared to regular software-based simulation, hardware emulation can reduce the time of fault injection experiments by more than four orders of magnitude [58, 62].

As we have seen, fault injection experiments are conducted at different levels of abstraction using a variety of tools. Thus, there is a need to conduct studies that compare the results obtained with different tools or at different levels of abstraction. In [64], the authors use simulation to compare the effects of bit-flip faults injected into the internal state elements of microprocessors with the effects observed when using pin-level fault injection and software-implemented fault injection (SWIFI) on the same processor. The results showed that SWIFI was able to mimic 98%–99% of the system-level error types observed for bit flips in the internal state elements. The corresponding numbers for the pin-level faults were only 9%–12%.

In [65], the authors compare the results of four fault injection techniques employed in the evaluation of a distributed real-time system. The results showed large variations in the impact of the fault manifestations observed among the techniques.

In a more recent study [30], the authors investigate whether simulations can provide accurate estimations of soft error rates (SER) for integrated circuits and program code. To this end, the authors compared SER estimations obtained by simulations and with results from neutron beam experiments. The study showed that SER estimations obtained by the two techniques were similar, which confirmed that simulations can indeed provide accurate estimations of soft error rates.

Another important line of research is to investigate uncertainty in measurements of dependability attributes. As pointed out by Bondavalli et al. [66], fault injection tools can be regarded as measurement instruments since they are used to measure dependability attributes. Another study related to uncertainty is [67], in which the authors investigated the metrological compatibility between results obtained with different fault injection techniques.

Recent work also includes the use of fault injection for emulating security attacks [68] and for evaluating safety-critical applications developed using articial intelligence and machine learning [69, 70]. There are also recent studies that utilise machine learning to assist in conducting fault injection experiments [71, 72]. A comprehensive survey of the use of software fault injection (SFI) is presented in [73].

1.3 Uncertainty vs factors of interest

Most types of measurements are affected by uncertainty. When we use measurements in scientific investigations, it is therefore essential to identify sources of uncertainty and, if possible, express quantitative bounds on the amount of uncertainty we expect to have in a measured value. Uncertainty is a key concept in the field of metrology, the science of measurement. General guidelines for dealing with and reporting uncertainty can be found in the Guide to the Expression of Uncertainty in Measurements (GUM), published by JCGM Joint committee for Guides in Metrology [74].

According to GUM, the word uncertainty can be used in two different senses: first, as a general concept meaning 'doubt about the validity of the results of a measurement', and second, as 'specific quantities that provide quantitative measures of the concept [sic: uncertainty]'. GUM provides examples of how uncertainty can be expressed quantitatively for measurements of physical quantities - for example, by calculating the mean and standard deviation for a series of repeated measurements.

In [66], Bondavalli et al. provide suggestions for how foundations in measurement theory can be applied in measurements of dependability attributes. The authors conclude that 'all sources of uncertainty should be clearly and univocally defined.' and that 'measurement uncertainty should be evaluated [according to the GUM].'

In the experiments presented in this thesis, we have identified two main sources of uncertainty. First, most of our experiments rely on sampling-based fault injections, as it would be too costly and time-consuming to perform exhaustive injections for most of the programs we evaluate.

Second, we have chosen not to control (initialise) the unused parts of the main memory of the target system. The reason for this decision is mainly practical: by not initialising the unused memory, we were able to reduce the time it took to conduct a fault-injection experiment. Further, we felt that leaving the memory uninitialised would be more representative of real working conditions than would be filling the unused areas with, for example, all 0s or all 1s. Of course, we could have chosen to fill the unused areas with a random pattern of 0s and 1s; however, the latter would resemble the situation we have when we do not initialise unused memory.

To quantify the level of uncertainty imposed by sampling, we relied on well-known statistical concepts, such as confidence intervals and hypothesis testing, as described in Section 1.5. Unfortunately, we had no means of quantifying the level of uncertainty that arose from choosing not to initialise unused memory. The construction of probabilistic models for predicting how variations in the content of unused memory influences measures of error sensitivity and other

dependability attributes is a research question that we were unable to address within the scope of this thesis.

Beyond the two sources of uncertainty discussed so far, there could have been other factors unknown to us that could have influenced the quality of our measurements. Such factors could have been hidden within the hardware or software units that were directly or indirectly involved in carrying out our experiments. Examples of such factors could be design defects (bugs) in the Goofi-2 software or the winIDEA debugger, which is used by Goofi-2.

Computer systems are in general complex systems; hence there are many factors that can influence the outcome of a fault injection experiment. We divide these factors into two categories: *sources of uncertainty* and *factors of interest*. A *source of uncertainty* is one that contributes to uncertainty in a measurement, as described above. A *factor of interest* is the object that we want to characterise with our measurements. A given aspect of a computer system can be either a source of uncertainty or a factor of interest depending on the objective of the measurements. For example, in our experiments we consider the content of unused memory a source of uncertainty. However, it would be possible to conduct experiments to investigate how the content of unused memory influences error sensitivity, in which case the content of unused memory would be the factor of interest.

In the following, unless otherwise stated, we use *factor* to denote a factor of interest. In general, we propose to divide such factors into three general types:

I Factors related to the operation of a system. Our study of the relationship between workload inputs and failure mode distributions presented in paper A belongs to this category. As previously mentioned, it would be of interest to conduct a study on how the content of unused memory or memory used by other programs, affects error sensitivity of a program. Such a study would belong to this category.

II Factors related to the design and implementation of a system. We investigate two factors in this category: compiler optimizations and source code implementations, which are described in paper B. There are many studies reported in the literature that belongs to this category. Most notably studies that evaluates error detection and fault tolerance mechanisms.

III Factors related to the assumption of the fault injection experiments. In this category, we investigate single vs double bit-flip errors in paper C, and the inject-on-read vs the inject-on-write technique in paper D. This category includes studies that address a fundamental problem in fault injection: selection and validation of fault models and tools.

1.4 Measuring Silent Data Corruptions

The goal of this thesis is to provide insight into how bit-flip errors in ISA registers and main memory words affect program execution. To this end, we conducted fault injection experiments using the Goofi-2 fault injection tool [75] with a collection of benchmark programs. The programs were executed on a target system consisting of a Freescale MPC565 microcontroller equipped with 512 KB of random-access memory. Goofi-2 uses test port—based fault

injection (i.e., the bit flips are injected using a hardware debugger connected to the microcontroller's test port).

The process of injecting a bit-flip error and recording the subsequent behaviour of the target program is fully automated. The Goofi-2 software runs on a PC that controls the operation of the target system and the hardware debugger. Goofi-2 supports two types of fault injection techniques: inject-on-write and inject-on-read. For inject-on-write, the bit-flip error is injected just after the target register/memory word has been updated by a machine instruction. For inject-on-read, the bit-flip error is injected just before the target register or memory word is read by a machine instruction.

Inject-on-write models the effects of bit flips that propagate into a register or memory word whereas inject-on-read models the effects of bit flips that occur while data is stored in a register or memory word. Inject-on-write and inject-on-read techniques ensure that bit flips are injected in locally live data. Thus, Goofi-2 avoids injecting errors into locally dead data words; however, a bit flip in a locally live data word will not necessarily cause the program to fail, as most programs perform redundant calculations that do not affect the output.

The impact of bit-flip errors in live data used by a program can be divided into four main categories: (i) detection by a machine exception, (ii) program hang, (iii) no effect and (iv) erroneous output with no error indication, also known as SDC. We focus on the SDCs since they are more costly to tolerate than program hangs, and errors detected by machine exceptions. Tolerating a program-level SDC requires the use of system-level fault-tolerance techniques that run programs redundantly.

Errors detected by machine instructions can be corrected by forward and backward recovery techniques. Program hangs (or delayed delivery of the output) can be detected by a watchdog timer and then corrected by a recovery action. Bit flips in locally live data often have no effect on the output of a program; thus, in addition to the three types of program failures, no effect is also a common outcome of our fault-injection experiments.

To characterise a program's tendency to exhibit an SDC due to a bit-flip error, we measure the proportion of bit-flip errors leading to an SDC versus all possible bit-flip errors that may occur in live data used by a program. We call this proportion the program's *error sensitivity*. Error sensitivity measures the occurrence of bit-flip errors in live data, and hence depends on the inputs processed by a program. (The number of live data items that a program creates and uses during execution depends on the inputs processed by the program.)

For some of the smaller programs, we were able to inject all possible single bit flips and thereby find the true value of the error sensitivity for the program for any input. However, for most of the programs, the total number of bit flips for any input were so large that we had to estimate the error sensitivity by sampling the population of all possible bit flips.

Error sensitivity is related to fault coverage, as both can be viewed as dependability measures; however, they serve different purposes. As mentioned in Section 1.2.1, fault coverage is the conditional probability that a system recovers and continues to deliver a correct service after a fault has occurred [76]. Thus, fault coverage is a measure of the effectiveness of a system's fault-tolerance capabilities. Fault coverage is therefore an important parameter when

estimating the reliability or availability of a fault-tolerant system.

In contrast, error sensitivity measures the tendency of a program to exhibit silent data corruptions. Its primary goal is to provide system designers with insights on how likely it is that a program will produce an SDC given the occurrence of a bit-flip error in a live data item. Measures of error sensitivity are therefore mainly aimed at guiding the system designer in selecting fault-tolerance techniques; for example, they can be used as a basis for a trade-off between software- and hardware-implemented fault-tolerance techniques.

We define error sensitivity as the proportion of test cases (bit-flip errors) in a test population that causes a program to produce an SDC. Note that the size of the test population depends on the inputs processed by the program.

We denote the error sensitivity of a given test population, T_i as es_i. In cases where we sample the test population, we use the following estimator for es_i:

$$\widehat{es_i} = \frac{s_i}{n_i} \tag{1.1}$$

where n_i is the number of faults (test cases) injected in fault injection campaign i, and s_i the number of faults that yielded an SDC.

As error sensitivity is a proportion, it may be problematic to use when comparing two programs that provide the same function but one of them is more efficient and has fewer machine instructions. Consider two programs a and b that provide the same function, where program a is more efficient and thus uses fewer machine instructions to implement the function than program b. Let $|T_a|$ and $|T_b|$ denote the size of the test population for program a and b, respectively, and let S_a and S_b denote the number of test cases that causes an SDC. We have that $|T_a| < |T_b|$, since program a uses fewer machine instructions than program b. However, it is possible that $S_a < S_b$ while $S_a/|T_a| > S_b/|T_b|$ ($S_i/|T_i|$ is the true error sensitivity for program i.) In others words, program a can have higher error sensitivity than program b, although the total number of test cases that lead to an SDC is lower for program a than for program b. Hence, comparing error sensitivities is not a good method for ranking the soft error vulnerability of programs. We therefore introduce SDC count as a suitable measure for comparing the reliability of programs with respect to soft errors.

We define *SDC count* as the number of test cases in T_i that causes the program to produce a *silent data corruption* (SDC). Let sc_i denote the SDC count. An intuitive estimator for sc_i is

$$\widehat{sc_i} = \frac{s_i}{n_i} |T_i| \tag{1.2}$$

where $|T_i|$ represents the cardinal (size) of T_i. Unfortunately, our fault injection tool, GOOFI-2 [75], does not provide a function that allows us to calculate $|T_i|$, although it would be technically feasible to implement such a function. Therefore, we use the following equation as an estimator for $|T_i|$

$$\widehat{|T_i|} = d_i \cdot m_i \tag{1.3}$$

where m_i is the number of machine instructions executed for a_i, and d_i is the average number of test cases per machine instruction in campaign i. If we

substitute $|T_i|$ with $\widehat{|T_i|}$ in equation (1.2), we obtain a less accurate estimator for sc_i, which we denote as $\widehat{sc_i^*}$

$$\widehat{sc_i^*} = \frac{s_i}{n_i} \cdot d_i \cdot m_i \tag{1.4}$$

Since we have reason to believe that the average number of test cases per machine instruction is fairly constant for similar programs, and indeed for different activities of one program, we simply assume a constant value for d_i. If we use $\widehat{sc_a^*}$ and $\widehat{sc_b^*}$ for ranking the reliability of programs a and b, then the values assigned to d_a and d_b will have no effect on the ranking result if they are assigned the same value. Hence, if we want to use SDC count estimates for only ranking purposes, then we can set the value of d_i in Equation 1.4 to 1. We can then view $\widehat{sc_i^*}$ as a strongly biased but consistent estimator of sc_i.

Considering our example with program a and b, where program a has a lower SDC count than program b, it is reasonable to assume that a has a lower failure probability but a higher error sensitivity than b. Hence, using the error sensitivity as a basis for ranking the reliability of the two programs would lead to an incorrect result. However, SDC count is an imperfect measure for ranking the reliability of different programs, since it does not consider the probability of occurrence for each test case $t \in |T|$. The problem associated with using measures based on proportions for comparing the soft error vulnerability of different programs has been addressed by Schirmeier et al. in [77].

So far, we have mainly discussed the estimation of the true values of es and sc for a single activity a for an executable program, and we have briefly compared different programs. To obtain a proper characterisation of the soft-error vulnerability of an executable program, though, we need to estimate es or sc for several activities (program inputs).

Aggregated measures of error sensitivity and SDC count can be achieved by calculating a weighted average of estimates obtained for the activities, where higher weights are given to activities that are expected to have a higher probability of occurrence. If data for determining the weight factors is unavailable, a common approach is simply to select an arbitrary set of activities and assign them equal weight factors. This is the approach we used in our experiments.

1.5 Statistical Analysis of Dependability Measurement

When conducting sampling-based fault injection experiments, we need to use statistical techniques for planning the experiments and assessing the uncertainty of estimated measures. Experimental design is a branch of statistics that deals with practical and theoretical aspects of experimental studies in three phases: (i) problem formulation, (ii) design of the experiments and (iii) analysis of the data collected [78]. In the first phase, the researcher states the problem and the purpose of the study, considering relevant studies and information in that field. In the second phase, the researcher chooses the response variables and anticipates influencing factors. The constraints, randomness mechanism, repeatability, number of samples and anticipated margin of error are aspects that should be considered in this phase. If the second phase is well designed,

then the third phase, the analysis of the collected data, is easy; indeed, the design phase determines which statistical method is to be used to analyse the collected data.

A series of studies [79, 80] were performed on stratified sampling for fault-injection experiments. These studies evaluated the importance of optimising the sampling method to increase the accuracy of the error-coverage estimation. Also, how defers to use frequentist or Bayesian approaches to estimate the measurand and identify the pros and cons of each approach.

Statistics also enable estimating the minimal number of experiments required to achieve a certain level of confidence and margin of error in the estimated measures. The authors of [81] show how to calculate the number of experiments depending on the error and confidence expectations. It may be difficult to determine the population size, or the possible fault space in a complex computer system. However, the researchers show that, when the population size exceeds 10,000, it has a little impact on the number of experiments or satisfying the margin of error.

The authors of [82] discussed a better approach to calculating sample size by (a) iteratively updating the sample size with more accurate assumptions of the population proportion by conducting few experiments (rather than considering 50% as a worst case assumption), and (b) considering the margin of error to be less than 5% (1% and even 0.1%) which is more sensible for some safety-critical applications.

In paper E, we provide a set of guidelines for selecting and applying statistical inference techniques for the analysis of data obtained in sampling-based fault injection experiments. We present methods for calculating confidence intervals and rules for determining when the normal approximation is applicable.

In the studies reported in paper A to D, we used the same statistical design of all fault injection campaigns. First, all our campaigns are designed to investigate a single factor of interest. Second, we decided to conduct 12,000 fault injection experiments in each campaign. As shown in paper E, this yields a worst case margin-of-error of 0.89 percentage points with a 95% confidence level, and a margin-of-error of 1.18 percentage points with a 99% confidence level. (The worst case margin-of-error occurs when the true value of the estimated value is 50%.) The reason for using 12,000 experiments was that we wanted to achieve confidence intervals of around ± 1 percentage point in our measurements.

In paper E, we also consider methods for hypothesis testing. We specifically discuss hypothesis testing for proportions, since error sensitivity is defined as a proportion. Hypothesis testing is especially useful when we want to compare two or more estimates of a dependability attribute obtained in different fault injection campaigns. We discuss various types of statistical tests for this purpose, including parametric and non-parametric tests.

Table 1.2 shows several statistical tests that are appropriate for comparing estimates obtained from two or more data sets. The choice of a test depends on circumstances such as the distribution of the data, whether the comparison is made for two or more data sets, and whether the data sets are paired or independent.

Table 1.2: Statistical tests for comparison

	2 data sets		More than 2 data sets	
Data Distribution	Paired data sets	Independent data sets	Paired data sets	Independent data sets
Normal Distribution	Paired Z-test (or T-test)	Unpaired Z-test	Repeated measures ANOVA	One-way ANOVA
Non-Normal Distribution	Wilcoxon Signed Rank	Mann-Whitney/ Wilcoxon rank sum	Friedman test	Kruskal-Wallis test
Categorical Distribution	McNemar's test	Chi-squared test/Fisher's exact test	Cochran's Q test	Chi-squared test

1.6 Papers and Contributions

Paper A. On the Impact of Hardware Faults — An Investigation of the Relationship between Workload Inputs and Failure Mode Distributions

Summary. This paper discusses the impact of input profile on a program's error sensitivity, how significant the impact is and how error sensitivity is correlated to input features. The variation of error sensitivity is application dependent. We found a linear correlation between input length and SDCs for some applications, while there were no correlations in other applications. In this study, we perform fault-injection experiments on four programs from MiBench suite. We selected nine inputs for each program. This study shows significant variation in the error sensitivity of a program executed with different inputs. For instance, in an extreme case, the error sensitivity of the CRC application varies by 30% from one input with 0 characters to another input with 99 characters.

In addition, we propose an approach to correlate the dynamic fault-free behaviour of a program with the error sensitivity. We observed significant variations in the error sensitivities among different workloads[1]. Hence, a program should be evaluated by all possible inputs. We propose a way to identify inputs that result in significantly different fault-injection outcomes. To this end, we use assembly metrics defined based on the machine instructions of a fault-free execution of a program. We cluster the workloads based on these assembly metrics and compare these clusters with those generated based on SDC outcomes. We discovered that workloads with similar SDC outcomes also have similar assembly metric clusters. Thus, the workloads that end up in the same cluster have similar SDC outcomes. In this way, we identify input sets that are likely to cause significantly different error sensitivities, which enables us to limit the number of necessary fault-injection campaigns to the number of clusters and perform only campaigns that generate significantly different error sensitivities.

[1] A *workload* is a program processing any input.

In this paper, we also evaluate the error sensitivity of the programs equipped with a software technique for tolerating hardware faults. This technique performs triple time-redundant execution and majority voting, and it can decrease the number of SDCs (on average, 13 times).

Statement of Contribution. This paper is co-authored with Domenico Di Leo, Behrooz Sangchoolie, Roger Johansson and Johan Karlsson. Domenico Di Leo had the original idea for the paper. Di Leo, Fatemeh Ayatolahi and Behrooz Sangchoolie jointly implemented and conducted the experiments, performed the data analysis and formulated the conclusions. They are also the main contributors in writing the paper. Johan Karlsson contributed to the idea and provided feedback during different phases of the study. Roger Johansson helped solve technical issues regarding the fault-injection setup. All authors contributed to the writing of the paper.

Paper B. A Study of the Impact of Bit-flip Errors on Programs Compiled with Different Optimization Levels

Summary. In Paper B, we investigate the impact of compiler optimizations on the error sensitivity of twelve benchmark programs. We conducted extensive fault-injection experiments in which bit-flip errors were injected in CPU registers and main-memory locations. The results show that the error sensitivity of the optimised programs is only marginally higher compared to that observed for the non-optimised programs. This suggests that compiler optimizations can be used in safety- and mission-critical systems without increasing the risk of the system producing undetected erroneous outputs. Program execution time reduces significantly by compiler optimization; therefore, the programs are also less exposed to transient faults.

Paper B also investigates the impact of different source-code implementations on the error sensitivity of functionally equivalent programs. To this end, we performed experiments on five bit-count programs included in the MiBench suite [83]. These programs differ in data types used to store results, using a look-up table for some pre-calculated values, and different ways of implementing calculations. The results of the fault-injection experiments show that source-code implementation significantly impacts error sensitivity. To provide insights into the reasons for the variation in the error sensitivity, we analysed the error sensitivity of different types of data stored in registers and memory words (that were targeted for fault injection). This analysis was helpful in identifying registers and memory sections with high error sensitivity, which are thus candidates for being protected by fault-tolerance techniques.

Statement of Contribution. This paper is co-authored with Behrooz Sangchoolie, Roger Johansson and Johan Karlsson. Fatemeh Ayatolahi and Behrooz Sangchoolie contributed to the original idea of the paper, the design and conduction of the fault injection experiments and the analysis of the results. They are also the main contributors in writing the paper. Johan Karlsson contributed to the idea and provided feedback during different phases of the study. Roger Johansson helped solve technical issues regarding the fault-injection setup. All authors contributed to the writing of the paper.

Paper C. A Study of the Impact of Single Bit-flip and Double Bit-flip Errors on Program Execution

Summary. This paper presents the results of an extensive experimental study of bit-flip errors in CPU registers and main-memory words. Comprising more than two million fault-injection experiments conducted with thirteen benchmark programs, the study provides insights on whether the double bit-flip model provides optimistic or pessimistic estimates of error sensitivity compared to the single bit-flip model. The results show that the proportion of errors that cause SDCs is almost the same for single- and double-bit errors. However, for some campaigns the single-bit errors resulted in a slightly higher proportion of SDCs, suggesting that single-bit errors on average tend to cause SDC more than double-bit errors.

In addition, we studied how error sensitivity varies for different bit positions within a register or memory word. We present detailed statistics about the variations in error sensitivity with respect to bit positions. The results show that error sensitivity varies significantly for different bit positions. An important observation is that injections in certain bit positions always have the same outcome, regardless of when the error is injected. For instance, all injections in more significant bit positions of the program counter register (PCR; e.g., bit positions 17 – 32) are detected by hardware exceptions.

Statement of Contribution. The paper is co-authored with Behrooz Sangchoolie, Roger Johansson and Johan Karlsson. Fatemeh Ayatolahi and Behrooz Sangchoolie contributed in the original idea of the paper, the design and conduction of the fault injection experiments and the analysis of the results. They are also the main contributors in writing the paper. Johan Karlsson contributed to the idea and provided feedback during different phases of the study. Roger Johansson helped solve technical issues regarding the fault-injection setup. All authors contributed to the writing of the paper.

Paper D. A Comparison of Inject-on-Read and Inject-on-Write in ISA-Level Fault Injection

Summary. In this paper, we compare two ISA-level fault-injection techniques to reduce the fault space and optimise the fault-injection experiments to discover SDCs. These two techniques are inject-on-read and inject-on-write. The inject-on-read technique injects bit flips into a data item just before it is read by a machine instruction, and inject-on-write injects bit flips into a data item just after it has been updated by a machine instruction. In other words, inject-on-read corrupts the content of the source register (or memory word) of a machine instruction, while inject-on-write corrupts the content of the destination register (or memory word) of a machine instruction. One advantage of these techniques is that the injections are made in conjunction with read and write operations, which ensures that the injected faults are always injected in (locally) live data items [84].

The inject-on-read and inject-on-write techniques can be used to emulate the effects of both hardware and software faults. However, in this paper we specifically consider the use of these techniques for assessing the hardware error sensitivity of programs and systems with respect to soft errors (i.e., particle-induced single-event upsets).

In this context, inject-on-read and inject-on-write target two classes of faults. Inject-on-read is well suited to model soft errors that occur in a data item during the time the item resides in an ISA register or memory word, while

inject-on-write is well suited for emulating errors that propagate into an ISA register or memory word. The first type of error occurs when an ISA register or memory word is hit directly by an ionising particle. The second type of error originates from particle strikes in other hardware resources within the microprocessor, such as ALUs, caches and internal pipeline registers. When using the inject-on-read technique for emulating direct strikes in ISA registers and main-memory words, a weight factor should be assigned to each injected fault. This weight factor should correspond to the length of the time interval during which a soft error can occur in reality.

This paper has two main objectives. The first is to compare the differences in impact between the techniques – specifically, whether one of the techniques is more likely to provoke SDCs than the other. The second objective is to investigate the impact of using weight factors for the inject-on-read technique. To this end, we compare the results of fault-injection experiments obtained with and without the use of such weight factors.

Our results are based on more than 120,000 inject-on-read and inject-on-write experiments with six programs from the automotive domain, including five implementations of the bit-count program included in the MiBench benchmark suite [83], and a prototype brake controller application.

Here, we observed that the error sensitivity obtained by the unweighted inject-on-read was higher than that of the inject-on-write. This indicates that, for the programs studied, it is unlikely that inject-on-write would expose weaknesses that are not revealed by the unweighted inject-on-read. This conclusion is drawn despite the significant differences between the source-code implementation of the programs under test.

The impact of the weight factor was studied by focusing on only the inject-on-read techniques. Here, we observed that, for all bit-count programs, the percentage of SDCs for the unweighted inject-on-read was higher than or equal to the percentage of SDCs for the weighted inject-on-read. However, this was not the case for the brake controller application, which shows that the weight factor influences the percentage of SDCs depending on the sensitivity of registers and memory words with long lifetimes. The difference between the SDC results obtained for these two techniques ranged from 0% to 20%.

Statement of Contribution. This paper is co-authored with Behrooz Sangchoolie, Roger Johansson and Johan Karlsson. Behrooz Sangchoolie contributed to the original idea of the paper, the design and conducting of the fault-injection experiments and the analysis of the results for the bit-count applications. Fatemeh Ayatolahi contributed to the design and implementation of the fault-injection experiments for the brake-by-wire application and provided feedback on the results. Johan Karlsson contributed by providing feedback during different phases of the study. Roger Johansson helped solve technical issues around the fault-injection setup. All authors contributed to the writing of the paper.

Paper E. Statistical Analysis of Fault-injection Data – A Case Study using Hypothesis Testing

Summary. In this paper, we summarise the impact of factors that affect error sensitivity estimation in fault-injection experiments. Such estimations are subject to sampling errors since they are calculated from a sample of all possible

fault scenarios. Hence, statistical inference techniques must be employed when drawing conclusions from such experiments. This paper presents a set of practical guidelines for selecting and applying statistical inference techniques in the analysis of data obtained from sampling-based fault-injection experiments. We used point estimation and confidence intervals to evaluate the margin of error for each measured proportion as the error sensitivity, and we used statistical hypothesis testing to analyse the variation in the estimated error sensitivities.

We considered three studies that use random sampling in fault-injection experiments. We considered two measures to characterise the soft-error vulnerability of a program. The aim was to assess the significance of the variations in the soft-error sensitivity and the SDC count for the executable programs in relation to (a) the inputs processed by the program, (b) the use of different levels of compiler optimization and (c) the injection of single bit-flip errors vs double bit-flip errors.

Statement of Contribution. This paper is co-authored with Johan Karlsson. Fatemeh Ayatolahi contributed to the original idea of the paper, performing the statistical inference and data analysis of the results, and is the main contributor in writing the paper. Johan Karlsson contributed to the idea and provided feedback during different phases of the study. Both authors contributed to the writing of the paper.

1.7 Concluding Remarks

This thesis presents the results of a series of fault injection experiments aimed at estimating the probability that a program exhibits a silent data corruption due to a bit-flip error in an ISA-register or main memory word holding live data. We use such bit-flips as an approximate model to investigate the impact transient hardware faults may have on program execution.

Our work is mainly motivated by the fact that modern integrated circuits are susceptible to soft errors, i.e., bit-flip errors caused by ionizing particles such as high-energy neutrons and alpha particles. However, bit-flips in ISA-registers and main memory words may also be considered as an approximate model for studying the impact of transistor failures caused by aging mechanisms such as NBTI (Negative Bias Temperature Instability) [85] and HCD (Hot Carrier Degradation) [86].

We introduce error sensitivity as a measure for the probability that a bit-flip error in an ISA-register or main memory location holding live data will cause a program to exhibit a silent data corruption. Our experiments show that the error sensitivity of a program is affected by different aspects of the program's design and use, and the set-up of the fault injection experiments. We denote these aspects as factors of variability and divide them into three groups: factors related to (i) the operation of the system, (ii) the design and implementation of the system, and (iii) the design of the fault injection experiments.

We have specifically conducted experiments to study variations in error sensitivity related to (i) the inputs processed by a program, (ii) the use of compiler optimization, (iii) source code implementation, (iv) single bit flips vs double bit flips and (v) injection techniques (inject-on-read vs inject-on-write).

Although our experiments provide valuable insights into how these factors influence the error sensitivity of a program, we would like to highlight some limitations of our work. Our fault models – single bit flips and double bit flips within one data word, with a uniform probability for all data words to be exposed to a soft error – are crude models of how soft errors occur in, or propagate into, ISA registers and main-memory words. For example, we have reason to believe that they do not accurately describe the true relative frequency with which soft errors affect live data in a program; the obvious reason for this is that a processor does not use the same hardware circuits to create all data words. Therefore, the probability of being exposed to a soft error varies for different data words.

It is also reasonable to assume that single bit flips occurring in the internals of a processor may manifest as multiple bit errors in ISA registers and main memory. For example, if a data word containing a single bit flip is used as input to an arithmetic operation, then the result of this operation may contain multiple bit flips. Another obvious limitation of our work is that some soft errors are likely to cause behaviours of microprocessors that simply cannot be modelled merely as bit flips in an ISA register or main-memory word.

An important task for future research is therefore to develop techniques that can accurately estimate the probability that a program exhibits a silent data corruption due to a soft error. However, the development of such techniques comes with both scientific and practical challenges. One such challenge is that this research would require full access to the hardware design of the microprocessor that will run the program, as well as detailed knowledge about the soft error rates of the integrated circuit technology used to produce the processor.

The correlation between different factors can be another interesting direction for future work. Researchers may investigate the correlations or dependencies between different factors that affect the error sensitivity, such as correlation that might be observed between input profile and different fault models. In our experiment design, we focused on one factor in each study to evaluate the impact of its variation on the estimated error sensitivities. These experiments can be extended to include the variation of multiple factors at a time to examine potential correlations between them. Consequently, careful experimental design should be planned, and other statistical inference techniques that analyse multiple factors should be considered. The authors of [87, 88] have introduced the usage of design of experiments and optimality methods in investigation of this interaction between different factors and find an optimal combination of them. The focus in these studies is on hardware synthesizing parameters and their interaction which is another level in system design which can be also applied in system implementation level.

Another aspect of our research is the comparison of the soft-error vulnerability between different programs. Here, we propose to use SDC count, rather than error sensitivity, as a measure for comparing the soft-error vulnerability between different programs. As pointed out by other researchers, measures such as error sensitivity, which are defined as proportions of outcomes of injected errors, are not suitable for comparing the reliability among different programs. This is because the execution time is a key factor in determining the probability that a program will be affected by a soft error. We propose a biased estimator

for the SDC count, which uses the execution time of a program to estimate the SDC count. A more accurate estimator could be achieved if the fault injection tool were provided with a function that could count all possible bit-flip errors in a program. We believe that implementing such a function would be easy, but it would be much more demanding to implement a function that would calculate the SDC count for a program. However, these are conjectures that we leave to be addressed by future research.

Bibliography

[1] A. Avižienis, J. . Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, Jan 2004.

[2] S. Borkar, "Designing reliable systems from unreliable components: the challenges of transistor variability and degradation," *IEEE Micro*, vol. 25, no. 6, pp. 10–16, 2005.

[3] J. F. Ziegler and W. A. Lanford, "Effect of cosmic rays on computer memories," *Science*, vol. 206, no. 4420, pp. 776–788, 1979. [Online]. Available: https://science.sciencemag.org/content/206/4420/776

[4] M. Čepin, *Reliability Block Diagram.* London: Springer London, 2011, pp. 119–123. [Online]. Available: https://doi.org/10.1007/978-0-85729-688-7_9

[5] E. Ruijters and M. Stoelinga, "Fault tree analysis," *Comput. Sci. Rev.*, vol. 15, no. C, pp. 29–62, Feb. 2015. [Online]. Available: http://dx.doi.org/10.1016/j.cosrev.2015.03.001

[6] J. R. Norris, *Markov Chains*, ser. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 1997.

[7] M. A. Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis, "Modelling with generalized stochastic petri nets," *SIGMETRICS Perform. Eval. Rev.*, vol. 26, no. 2, p. 2, Aug. 1998. [Online]. Available: https://doi.org/10.1145/288197.581193

[8] W. H. Sanders and J. F. Meyer, *Stochastic Activity Networks: Formal Definitions and Concepts.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 315–343. [Online]. Available: https://doi.org/10.1007/3-540-44667-2_9

[9] M. Wallace, "Modular architectural representation and analysis of fault propagation and transformation," *Electron. Notes Theor. Comput. Sci.*, vol. 141, no. 3, p. 5371, Dec. 2005. [Online]. Available: https://doi.org/10.1016/j.entcs.2005.02.051

[10] Z. Haider, B. Gallina, and E. Z. Moreno, "Fla2ft: Automatic generation of fault tree from concertofla results," in *2018 3rd International Conference on System Reliability and Safety (ICSRS)*, 2018, pp. 176–181.

[11] F. Mhenni, N. Nguyen, and J. Choley, "Automatic fault tree generation from sysml system models," in *2014 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, 2014, pp. 715–720.

[12] I. Šljivo, B. Gallina, J. Carlson, H. Hansson, and S. Puri, "A method to generate reusable safety case argument-fragments from compositional safety analysis," *Journal of Systems and Software*, vol. 131, pp. 570–590, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121216301273

[13] M. Kwiatkowska, G. Norman, and D. Parker, *Probabilistic Model Checking: Advances and Applications*. Cham: Springer International Publishing, 2018, pp. 73–121. [Online]. Available: https://doi.org/10.1007/978-3-319-57685-5_3

[14] E. M. Hart and K. Bell, *prism: Download data from the Oregon prism project*, 2015, r package version 0.0.6. [Online]. Available: http://github.com/ropensci/prism

[15] O. Hasan, S. Tahar, and N. Abbasi, "Formal reliability analysis using theorem proving," *IEEE Transactions on Computers*, vol. 59, no. 5, pp. 579–592, May 2010.

[16] W. Ahmad and O. Hasan, "Formalization of fault trees in higher-order logic: A deep embedding approach," in *Dependable Software Engineering: Theories, Tools, and Applications*, M. Fränzle, D. Kapur, and N. Zhan, Eds. Cham: Springer International Publishing, 2016, pp. 264–279.

[17] R. K. Sahoo, M. S. Squillante, A. Sivasubramaniam, and Y. Zhang, "Failure data analysis of a large-scale heterogeneous server environment," in *International Conference on Dependable Systems and Networks, 2004*, June 2004, pp. 772–781.

[18] B. Schroeder and G. Gibson, "A large-scale study of failures in high-performance computing systems," *IEEE Trans. Dependable Secur. Comput.*, vol. 7, no. 4, pp. 337–351, Oct. 2010. [Online]. Available: http://dx.doi.org/10.1109/TDSC.2009.4

[19] J. A. Duraes and H. S. Madeira, "Emulation of software faults: A field data study and a practical approach," *IEEE Transactions on Software Engineering*, vol. 32, no. 11, pp. 849–867, 2006.

[20] J. Christmansson and R. Chillarege, "Generation of an error set that emulates software faults based on field data," in *Proceedings of Annual Symposium on Fault Tolerant Computing*, 1996, pp. 304–313.

[21] J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J. . Fabre, J. . Laprie, E. Martins, and D. Powell, "Fault injection for dependability validation: a methodology and some applications," *IEEE Transactions on Software Engineering*, vol. 16, no. 2, pp. 166–182, Feb 1990.

[22] H. Madeira, M. Rela, F. Moreira, and J. G. Silva, "Rifle: A general purpose pin-level fault injector," in *Dependable Computing — EDCC-1*, K. Echtle,

D. Hammer, and D. Powell, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, pp. 197–216.

[23] U. Gunneflo, J. Karlsson, and J. Torin, "Evaluation of error detection schemes using fault injection by heavy-ion radiation," in *[1989] The Nineteenth International Symposium on Fault-Tolerant Computing. Digest of Papers*, 1989, pp. 340–347.

[24] J. Arlat, Y. Crouzet, and J. . Laprie, "Fault injection for dependability validation of fault-tolerant computing systems," in *[1989] The Nineteenth International Symposium on Fault-Tolerant Computing. Digest of Papers*, 1989, pp. 348–355.

[25] J. Karlsson, P. Liden, P. Dahlgren, R. Johansson, and U. Gunneflo, "Using heavy-ion radiation to validate fault-handling mechanisms," *IEEE Micro*, vol. 14, no. 1, pp. 8–23, 1994.

[26] J. Karlsson, J. Arlat, and G. Leber, "Application of three physical fault injection techniques to the experimental assessment of the mars architecture," in *1995, Proceeding, Fifth Ann. IEEE Int'l Working Conf. Dependable Computing for Critical Applications*, 1995, pp. 150–161.

[27] L. Claudepierre and P. Besnier, "Microcontroller sensitivity to fault-injection induced by near-field electromagnetic interference," in *2019 Joint International Symposium on Electromagnetic Compatibility, Sapporo and Asia-Pacific International Symposium on Electromagnetic Compatibility (EMC Sapporo/APEMC)*, 2019, pp. 673–676.

[28] J. Samson, W. Moreno, and F. Falquez, "Validating fault tolerant designs using laser fault injection (lfi)," in *1997 IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, 1997, pp. 175–183.

[29] D. H. Habing, "The use of lasers to simulate radiation-induced transients in semiconductor devices and circuits," *IEEE Transactions on Nuclear Science*, vol. 12, no. 5, pp. 91–100, 1965.

[30] A. Chatzidimitriou, P. Bodmann, G. Papadimitriou, D. Gizopoulos, and P. Rech, "Demystifying soft error assessment strategies on arm cpus: Microarchitectural fault injection vs. neutron beam experiments," in *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2019, pp. 26–38.

[31] S. Han, K. Shin, and H. Rosenberg, "Doctor: an integrated software fault injection environment for distributed real-time systems," in *Proceedings of 1995 IEEE International Computer Performance and Dependability Symposium*, 1995, pp. 204–213.

[32] G. Kanawati, N. Kanawati, and J. Abraham, "Ferrari: a flexible software-based fault and error injection system," *IEEE Transactions on Computers*, vol. 44, no. 2, pp. 248–260, 1995.

[33] H. Schirmeier, M. Hoffmann, R. Kapitza, D. Lohmann, and O. Spinczyk, "Fail∗: Towards a versatile fault-injection experiment framework," in *ARCS 2012*, 2012, pp. 1–5.

[34] J. Carreira, H. Madeira, and J. G. Silva, "Xception: a technique for the experimental evaluation of dependability in modern computers," *IEEE Transactions on Software Engineering*, vol. 24, no. 2, pp. 125–136, 1998.

[35] J. Barton, E. Czeck, Z. Segall, and D. Siewiorek, "Fault injection experiments using fiat," *IEEE Transactions on Computers*, vol. 39, no. 4, pp. 575–582, 1990.

[36] D. Stott, B. Floering, D. Burke, Z. Kalbarczpk, and R. Iyer, "Nftape: a framework for assessing dependability in distributed systems with lightweight fault injectors," in *Proceedings IEEE International Computer Performance and Dependability Symposium. IPDS 2000*, 2000, pp. 91–100.

[37] P. Yuste, D. de Andres, L. Lemus, J. Serrano, and P. Gil, "Inerte: integrated nexus-based real-time fault injection tool for embedded systems," in *2003 International Conference on Dependable Systems and Networks, 2003. Proceedings.*, 2003, pp. 669–669.

[38] P. Yuste, J. C. Ruiz, L. Lemus, and P. Gil, "Non-intrusive software-implemented fault injection in embedded systems," in *Dependable Computing*, R. de Lemos, T. S. Weber, and J. B. Camargo, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 23–38.

[39] J.-C. Ruiz, J. Pardo, J.-C. Campelo, and P. Gil, "On-chip debugging-based fault emulation for robustness evaluation of embedded software components," in *11th Pacific Rim International Symposium on Dependable Computing (PRDC'05)*, 2005, pp. 8 pp.–.

[40] J. Aidemark, J. Vinter, P. Folkesson, and J. Karlsson, "Goofi: generic object-oriented fault injection tool," in *2001 International Conference on Dependable Systems and Networks*, 2001, pp. 83–88.

[41] P. Folkesson, S. Svensson, and J. Karlsson, "A comparison of simulation based and scan chain implemented fault injection," in *Digest of Papers. Twenty-Eighth Annual International Symposium on Fault-Tolerant Computing (Cat. No.98CB36224)*, 1998, pp. 284–293.

[42] M. Liu, Z. Zeng, F. Su, and J. Cai, "Research on fault injection technology for embedded software based on jtag interface," in *2016 11th International Conference on Reliability, Maintainability and Safety (ICRMS)*, 2016, pp. 1–6.

[43] H. Madeira, D. Costa, and M. Vieira, "On the emulation of software faults by software fault injection," in *Proceeding International Conference on Dependable Systems and Networks. DSN 2000*, 2000, pp. 417–426.

[44] J. Duraes and H. Madeira, "Emulation of software faults by educated mutations at machine-code level," in *13th International Symposium on Software Reliability Engineering, 2002. Proceedings.*, 2002, pp. 329–340.

[45] R. Chillarege and N. Bowen, "Understanding large system failures-a fault injection experiment," in *[1989] The Nineteenth International Symposium on Fault-Tolerant Computing. Digest of Papers*, 1989, pp. 356–363.

[46] W.-L. Kao and R. Iyer, "Define: a distributed fault injection and monitoring environment," in *Proceedings of IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems*, 1994, pp. 252–259.

[47] Z. Kazemi, A. Papadimitriou, I. Souvatzoglou, E. Aerabi, M. M. Ahmed, D. Hely, and V. Beroulle, "On a low cost fault injection framework for security assessment of cyber-physical systems: Clock glitch attacks," in *2019 IEEE 4th International Verification and Security Workshop (IVSW)*, 2019, pp. 7–12.

[48] (2003) NEXUS debugger. [Online]. Available: http://nexus5001.org/wp-content/uploads/2015/02/APB179-NexusBooklet-1.pdf

[49] (1995) winIDEA integrated development environment. [Online]. Available: https://www.isystem.com/products/software/winidea.html

[50] K. Parasyris, G. Tziantzoulis, C. D. Antonopoulos, and N. Bellas, "Gemfi: A fault injection tool for studying the behavior of applications on unreliable substrates," in *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2014, pp. 622–629.

[51] E. Touloupis, J. A. Flint, V. A. Chouliaras, and D. D. Ward, "Study of the effects of seu-induced faults on a pipeline protected microprocessor," *IEEE Transactions on Computers*, vol. 56, no. 12, pp. 1585–1596, 2007.

[52] G. Choi and R. Iyer, "Focus: an experimental environment for fault sensitivity analysis," *IEEE Transactions on Computers*, vol. 41, no. 12, pp. 1515–1526, 1992.

[53] R. Svenningsson, J. Vinter, H. Eriksson, and M. Törngren, "Modifi: A model-implemented fault injection tool," in *Computer Safety, Reliability, and Security*, E. Schoitsch, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 210–222.

[54] S. K. S. Hari, S. V. Adve, H. Naeimi, and P. Ramachandran, "Relyzer: Exploiting application-level fault equivalence to analyze application resiliency to transient faults," in *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS XVII. New York, NY, USA: Association for Computing Machinery, 2012, p. 123134. [Online]. Available: https://doi.org/10.1145/2150976.2150990

[55] K. Goswami, "Depend: a simulation-based environment for system level dependability analysis," *IEEE Transactions on Computers*, vol. 46, no. 1, pp. 60–74, 1997.

[56] E. Jenn, J. Arlat, M. Rimen, J. Ohlsson, and J. Karlsson, "Fault injection into vhdl models: the mefisto tool," in *Proceedings of IEEE 24th International Symposium on Fault- Tolerant Computing*, 1994, pp. 66–75.

[57] G. Saggese, N. Wang, Z. Kalbarczyk, S. Patel, and R. Iyer, "An experimental study of soft errors in microprocessors," *IEEE Micro*, vol. 25, no. 6, pp. 30–39, 2005.

[58] P. Civera, L. Macchiarulo, M. Rebaudengo, M. Reorda, and M. Violante, "Exploiting circuit emulation for fast hardness evaluation," *IEEE Transactions on Nuclear Science*, vol. 48, no. 6, pp. 2210–2216, 2001.

[59] D. de Andres, J. C. Ruiz, D. Gil, and P. Gil, "Fault emulation for dependability evaluation of vlsi systems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 4, pp. 422–431, 2008.

[60] ——, "Fades: a fault emulation tool for fast dependability assessment," in *2006 IEEE International Conference on Field Programmable Technology*, 2006, pp. 221–228.

[61] R. Leveugle and A. Prost-Boucle, "A new automated instrumentation for emulation-based fault injection," in *2010 First IEEE Latin American Symposium on Circuits and Systems (LASCAS)*, 2010, pp. 200–203.

[62] A. Ejlali and S. G. Miremadi, "Error propagation analysis using fpga-based seu-fault injection," *Microelectronics Reliability*, vol. 48, no. 2, pp. 319–328, 2008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0026271407001746

[63] M. Ebrahimi, A. Mohammadi, A. Ejlali, and S. G. Miremadi, "A fast, flexible, and easy-to-develop fpga-based fault injection technique," *Microelectronics Reliability*, vol. 54, no. 5, pp. 1000–1008, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0026271414000067

[64] M. Rimen, J. Ohlsson, and J. Torin, "On microprocessor error behavior modeling," in *Proceedings of IEEE 24th International Symposium on Fault-Tolerant Computing*, 1994, pp. 76–85.

[65] J. Arlat, Y. Crouzet, J. Karlsson, P. Folkesson, E. Fuchs, and G. Leber, "Comparison of physical and software-implemented fault injection techniques," *IEEE Transactions on Computers*, vol. 52, no. 9, pp. 1115–1133, 2003.

[66] A. Bondavalli, A. Ceccarelli, L. Falai, and M. Vadursi, "Foundations of measurement theory applied to the evaluation of dependability attributes," in *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, June 2007, pp. 522–533.

[67] D. Skarin, R. Barbosa, and J. Karlsson, "Comparing and validating measurements of dependability attributes," in *2010 European Dependable Computing Conference*, April 2010, pp. 3–12.

[68] B. Sangchoolie, P. Folkesson, P. Kleberger, and J. Vinter, "Analysis of cybersecurity mechanisms with respect to dependability and security attributes," in *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, 2020, pp. 94–101.

[69] K. Pattabiraman, G. Li, and Z. Chen, "Error resilient machine learning for safety-critical systems: Position paper," in *2020 IEEE 26th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2020, pp. 1–4.

[70] D. Cotroneo, L. De Simone, P. Liguori, and R. Natella, "Fault injection analytics: A novel approach to discover failure modes in cloud-computing systems," *IEEE Transactions on Dependable and Secure Computing*, 2020, early access.

[71] M. Moradi, B. J. Oakes, and J. Denil, "Machine Learning-assisted Fault Injection," in *39th International Conference on Computer Safety, reliability and Security (SAFECOMP), Position Paper, Lisbon, Portugal*, Lisbon, Portugal, Sep. 2020. [Online]. Available: https://hal.laas.fr/hal-02931709

[72] S. Jha, S. Banerjee, T. Tsai, S. K. S. Hari, M. B. Sullivan, Z. T. Kalbarczyk, S. W. Keckler, and R. K. Iyer, "Ml-based fault injection for autonomous vehicles: A case for bayesian fault injection," in *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, June 2019, pp. 112–124.

[73] R. Natella, D. Cotroneo, and H. S. Madeira, "Assessing dependability with software fault injection: A survey," *ACM Comput. Surv.*, vol. 48, no. 3, Feb. 2016. [Online]. Available: https://doi.org/10.1145/2841425

[74] J. C. for Guides in Metrology, "Jcgm 100: Evaluation of measurement data guide to the expression of uncertainty in measurement," JCGM, Tech. Rep., 2008.

[75] D. Skarin, R. Barbosa, and J. Karlsson, "Goofi-2: A tool for experimental dependability assessment," in *2010 IEEE/IFIP International Conference on Dependable Systems Networks (DSN)*, 2010, pp. 557–562.

[76] W. G. Bouricius, W. C. Carter, and P. R. Schneider, "Reliability modeling techniques for self-repairing computer systems," in *Proceedings of the 1969 24th National Conference*, ser. ACM '69. New York, NY, USA: ACM, 1969, pp. 295–309. [Online]. Available: http://doi.acm.org/10.1145/800195.805940

[77] H. Schirmeier, C. Borchert, and O. Spinczyk, "Avoiding pitfalls in fault-injection based comparison of program susceptibility to soft errors," in *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, June 2015, pp. 319–330.

[78] J. S. Milton and J. C. Arnold, *Introduction to Probability and Statistics: Principles and Applications for Engineering and the Computing Sciences*, 4th ed. USA: McGraw-Hill, Inc., 2002.

[79] M. Cukier, D. Powell, and J. Ariat, "Coverage estimation methods for stratified fault-injection," *IEEE Transactions on Computers*, vol. 48, no. 7, pp. 707–723, July 1999.

[80] D. Powell, E. Martins, J. Arlat, and Y. Crouzet, "Estimators for fault tolerance coverage evaluation," in *FTCS-23 The Twenty-Third International Symposium on Fault-Tolerant Computing*, June 1993, pp. 228–237.

[81] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert, "Statistical fault injection: Quantified error and confidence," in *2009 Design, Automation Test in Europe Conference Exhibition*, April 2009, pp. 502–506.

[82] I. Tuzov, D. de Andrs, and J. Ruiz, "Accurate robustness assessment of hdl models through iterative statistical fault injection," in *2018 14th European Dependable Computing Conference (EDCC)*, 2018, pp. 1–8.

[83] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization. WWC-4 (Cat. No.01EX538)*, Dec 2001, pp. 3–14.

[84] R. Barbosa, J. Vinter, P. Folkesson, and J. Karlsson, "Assembly-level pre-injection analysis for improving fault injection efficiency," in *Dependable Computing - EDCC 5*, M. Dal Cin, M. Kaâniche, and A. Pataricza, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 246–262.

[85] H. Hong, J. Lim, H. Lim, and S. Kang, "Lifetime reliability enhancement of microprocessors: Mitigating the impact of negative bias temperature instability," *ACM Comput. Surv.*, vol. 48, no. 1, Sep. 2015. [Online]. Available: https://doi.org/10.1145/2785988

[86] B. Ullmann, M. Jech, K. Puschkarsky, G. A. Rott, M. Waltl, Y. Illarionov, H. Reisinger, and T. Grasser, "Impact of mixed negative bias temperature instability and hot carrier stress on mosfet characteristicspart i: Experimental," *IEEE Transactions on Electron Devices*, vol. 66, no. 1, pp. 232–240, 2019.

[87] I. Tuzov, D. d. Andrs, and J.-C. Ruiz, "Dependability-aware design space exploration for optimal synthesis parameters tuning," in *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2017, pp. 121–132.

[88] I. Tuzov, D. de Andrs, and J.-C. Ruiz, "Robustness-aware design space exploration through iterative refinement of d-optimal designs," in *2019 15th European Dependable Computing Conference (EDCC)*, 2019, pp. 23–30.