Master Thesis

Prototyping an efficient and cost-effective method to detect and mitigate random faults in COTS processors

Konstantinos Karavelas

Delft University of Technology

TU Delft

Master Thesis

Prototyping an efficient and cost-effective method to detect and mitigate random faults in COTS processors

by

Konstantinos Karavelas

ET4300 - Master Thesis,

for a Double MSc. Degree in Aerospace Engineering and Computer Engineering

To be publicly defended on November 3rd, 2020 at 14.30

Delft University of Technology

ARM Limited, Cambridge

Student number:	4508777	
Thesis number:	Q&CE-CE-MS-2020-11	
Project duration:	January - October, 2020	
TU Delft Supervisors:	Prof. dr. ir. A. Menicucci,	Faculty of AE
	Prof. dr. ir. A.J. van Genderen	Faculty of EEMCS
ARM Supervisors:	M. Lahori,	Principal Engineer
	I. D. Prima	Staff Engineer

Cover Photo: Rendezvous Autonomous CubeSats Experiment (RACE). Credits ESA [31].

Preface

In 2015, while going through a very stressful period in my life, I received an email from TU Delft's admission office that I was admitted to the M.Sc. programme in Computer Engineering. This filled me with joy and a feeling of accomplishment, looking forward to the start of a new chapter in my life.

What came next was an exciting couple of months, enjoying life as a student while meeting lots of new people, studying, partying, traveling and working hard to take advantage of everything that the multicultural community of TU Delft had to offer. Upon completing the first year of courses, I decided to apply for a Double M.Sc. Degree with Aerospace Engineering, in an effort to combine my passion for computers and space and embark on a quest for knowledge.

Fast forward to today, I would never have imagined what would happen in the years to come and how these would shape me as a person. I am grateful for all the intriguing challenges, the failures and successes and the valuable lessons learned, sometimes the hard way, while daring to do things that I would never have imagined before. They helped me become a better individual and brought me to where I am today. After five exciting and memorable years, I am almost ready to graduate.

This moment would not have been possible without the help of certain individuals. More specifically, I would like to express my gratitude to my university supervisors, Alessandra Menicucci and Arjan van Genderen for their help and support the past year, for being flexible and devoting the time to resolve issues. In addition, I would also like to thank Ross Torkington for giving me the chance to join the company and perform my M.Sc. thesis project at the same time. I will be forever grateful to him. Furthermore, I would like to thank Mukesh Lahori and Ivan Di Prima for their supervision and patience, especially during the first months of the project and for creating a friendly working environment.

Finally, I would like to thank my parents and Despoina, for their unconditional love and for believing in me even in times when I doubted myself.

Konstantinos Karavelas
Cambridge, October 2020

Contents

List of Figures iii

List of Tables iv

1 Introduction 1
- 1.1 Background information 1
- 1.2 Introduction to the thesis 2
- 1.3 ARM Limited: The Company 2
- 1.4 Software Test Libraries team 2

2 Research Framework 4
- 2.1 Research Questions 4
- 2.2 Sources of radiation 5
- 2.3 Classes of radiation effects 5
- 2.4 Single Event Effects (SEE) on CMOS devices 6
  - 2.4.1 Single Event Upset (SEU) 7
  - 2.4.2 Single Event Transient (SET) 7
  - 2.4.3 Single Event Functional Interrupt (SEFI) 7
  - 2.4.4 Single Event Latch-up (SEL) 8
  - 2.4.5 Single Event Burnout (SEB)& Single Event Gate Rupture (SEGR) 8
- 2.5 Random Error Detection & Mitigation 9
  - 2.5.1 Device Level 9
  - 2.5.2 Microarchitecture Level 10
  - 2.5.3 System Level 11
- 2.6 Mitigation in popular COTS processors 12
- 2.7 Novel technologies and Reconfigurable Computing. 12
- 2.8 Conclusions. 13
  - 2.8.1 Radiation effects 13
  - 2.8.2 Current mitigation techniques 14

3 Problem Statement 15
- 3.1 Problem Statement 15
- 3.2 Purpose Statement 15
- 3.3 Restrictions 16

4 Methodology 17
- 4.1 Overview: STL architecture 17
- 4.2 Reference platform 18
  - 4.2.1 Subsystem selection 19
- 4.3 STL Design 20
  - 4.3.1 AXI Interface part 20
  - 4.3.2 Load Unit 21
  - 4.3.3 Store Unit 21
  - 4.3.4 Hazarding Logic 21
- 4.4 Verification Process 22
  - 4.4.1 Verification Design. 23
  - 4.4.2 Regressions 24

5 Orbit & Fault Simulation 25
- 5.1 Fault Modeling 25
  - 5.1.1 Permanent faults. 25
  - 5.1.2 Transient faults 26

5.2 SPENVIS simulations 27
5.2.1 LEO reference missions 27
5.2.2 MEO-GEO reference missions 29
5.3 Conclusions. 31

6 Results 32
6.1 Fault Coverage 32
6.1.1 Permanent faults. 32
6.1.2 Transient faults 33
6.2 Regression results 35
6.3 Porting to a new Revision 36

7 Conclusion 37
7.1 Objective statement. 37
7.2 Achieved outcome 37
7.3 Limitations 38

8 Future Work 40
8.1 Improving existing framework 40
8.2 Physical tests 41

A Appendix 43

Bibliography 44

List of Figures

2.1 Illustrations of the three radiation sources. 5
2.2 SEE mechanisms of interaction with silicon devices from [24]. 6
2.3 SEU and SET relationship with device and LET scaling. 8
2.4 Examples of SEL damage and underlying mechanism. 9
2.5 The underlying mechanisms of SEB and SEGR effects. 9
2.6 Two prominent examples of successful device-level error mitigation. 10
2.7 Breakdown of spacecraft anomalies from NASA's study [11], as presented in [26]. 13

4.1 The STL framework, which acts as the starting point of this work. 18
4.2 Overview of the reference platform used for STL development during the thesis. 19
4.3 The STL Verification Process, used in the scope of this work. 22

5.1 Locations of stuck-at-fault placement. 26
5.2 LEOs are primarily used for Earth Observation space missions or science missions using CubeSats or similar low-mass, low-volume spacecrafts. A characteristic example of such mission is the Delfi-C3 built from COTS. 28
5.3 Communications satellites use GEO staying constantly above a specific point on Earth. Navigation satellites use somewhat lower orbits (MEO) to provide coverage across large parts of the world. 29

6.1 Transient fault simulation results. 34

8.1 Examples of radiation testing efforts on COTS processors. 41

List of Tables

5.1 Simulated radiation upset rates for the selected LEO missions, using SPENVIS. A width of 0.1 cm of shielding was assumed for the electronics compartment. Items with an asterisk (*) are best-case estimations based on available orbit data, whereas items with a dagger (†) represent a generic orbit to be used for reference. 29
5.2 Simulated radiation upset rates for the selected MEO-GEO missions, using SPENVIS. A width of 0.1 cm of shielding was assumed for the electronics compartment. Items with an asterisk (*) are best-case estimations based on available data, whereas items with a dagger (†) represent a generic orbit to be used for reference. Finally items with a double dagger (‡) represent information for the selected GSAT0204 Galileo mission. 30

6.1 Fault simulation results for the major modules only, using ZOIX. Testable faults are listed as Dropped Detected (D-D), Not Detected (N-D) and Detected stop/finish (D-F). Untestable faults are not presented, since they do not contribute to the total fault coverage. A more detailed view of the fault simulation results can be seen in Section 5 of the Detailed Design Report. 33
6.2 Status of the verification flow, listing the supported features and functionality. Simulations were performed with Questa Advanced Simulator. 35

A.1 Overview of the most popular microprocessors, for the selected CubeSat missions [40]. Entries with a (*) are best-case estimates based on memory mapping, whereas entries with a (†) are model dependent. 43
A.2 Overview of the most popular radiation-hardened microprocessors. Entries with a (*) are best-case estimates based on internal memory mapping, whereas entries with a (†) depend on model version. 43
A.3 Radiation performance of all selected processors. Entries with a (†) depend on model version, whereas dash lines denote no available data. A (‡) indicates classification based on information from [35, 66]. 43

Acronyms

AGC Apollo Guidance Computer. 1

AI Artificial Intelligence. 18, 19

API Application Programming Interface. 17

ASIL Automotive Safety Integrity Level. 2, 17, 22, 40

AXI Advanced eXtensible Interface. 19–21, 32, 33, 37

BIRA-IASB Royal Belgian Institute for Space Aeronomy. 27

BIST Built-In Self-Test. 17, 25

BJT Bipolar Junction Transistor. 8–10, 25

CCD Charge-Coupled Devices. 6

CMOS Complementary Metal-Oxide-Semiconductor. 4, 7–9, 13, 25, 31, 37

CNES Centre national d'études spatiales (National Centre for Space Studies). 28

COTS Commercial Off-The-Shelf components. iii, ix, 1, 2, 4, 12–16, 25, 26, 28, 37–41

CPU Central Processing Unit. ix, 2–4, 11–19, 22, 25, 26, 28, 31, 33, 36–41, 44

CRC Cyclic Redundancy Checks. 10

CubeSat Miniaturized satellite for space research that is made up of multiples of 10 cm × 10 cm × 10 cm cubic units. i, iii, ix, 1, 2, 12, 28, 40

D-cache Data Cache. 20, 21

D-D Dropped Detected. iv, 32, 33

D-F Detected stop/finish. iv, 32, 33

DD Displacement Damage. 6, 17, 25

Delfi-C3 Delfi-C3 CubeSat mission from TU Delft. iii, 28

DICE Dual Interlocked storage Cell. 10

DMOS Double-diffused Metal-Oxide-Semiconductor. 8, 9

DRAM Dynamic Random-Access Memory. 7, 43

DUT Device Under Test. 41, 42

ECC Error Correction Codes. 10, 26, 38

ECU Engine Control Unit. 2

EDAC Error Detection And Correction. 10–14

EO Earth Observation space mission. iii, 27, 28

ESA European Space Agency. i, 27–29

FIFO First In, First Out. 20

FPGA Field-Programmable Gate Array. 7, 13, 15, 40, 41, 43

FSM Finite State Machine. 7

Galileo Galileo, ESA's global navigation system. 29, 30

GEO Geosynchronous Equatorial Orbit. iii, iv, ix, 29, 30, 33, 41

GPS Global Positioning System. 29, 30

GPU Graphics Processing Unit. 2

I-cache Instruction Cache. 20

I2C Inter-Integrated Circuit. 11

IC Integrated Circuit. 1, 4, 6–8, 11, 13, 37

IEC 61508 International functional safety standard. 26, 32

IP Intellectual Property. 2, 39, 40

ISA Instruction Set Architecture. 2, 13, 40

ISO 26262 Automotive functional safety standard. 2, 17, 22, 26, 32

LEO Low Earth Orbit. iii, iv, ix, 1, 2, 27–30, 33, 38, 41

LET Linear Energy Transfer. iii, 6–8, 41

MBIST Memory Built-In Self-Test. 25

MEO Medium Earth Orbit. iii, iv, 29, 30

MIU Main Interface Unit. ix, 19–21, 27, 28, 31–33, 36–38, 40

MOSFET Metal-Oxide-Semiconductor Field-Effect Transistor. 5, 6, 8, 25

N-D Not Detected. iv, 32, 33

NASA National Aeronautics and Space Administration. 12, 28

NEPP NASA Electronic Parts and Packaging program. 12

NMOS N-channel Metal-Oxide-Semiconductor Field-Effect Transistor. 6, 8, 10

NoC Network-on-Chip. 19

PCB Printed Circuit Board. 11

PL Programmable Logic. 13

PMOS P-channel Metal-Oxide-Semiconductor Field-Effect Transistor. 8, 10

Questa Questa Advanced Simulator from Mentor. iv, 23, 35

RAR Read-After-Read hazard. 20

RAW Read-After-Write hazard. 20, 21

RISC Reduced Instruction Set Computer. 2, 43

RO Read-Only register. 23

RTL Register-Transfer Level layer in computing abstraction. 35, 36, 39, 41

RTOS Real-Time Operating System. ix, 17, 22, 33

RW Read/Write register. 23, 35

SEB Single Event Burnout. iii, 8, 9

SEE Single Event Effects. iii, viii, 6–8, 11, 13, 25, 26, 41

SEFI Single Event Functional Interrupt. 7, 11, 26

SEGR Single Event Gate Rupture. iii, 8, 9, 25

SEL Single Event Latch-up. iii, 8–10, 12, 13, 25

SEM Scanning Electron Microscopy. 9

SET Single Event Transient. iii, 7, 8, 11, 13, 17, 26, 27

SEU Single Event Upset. iii, 7, 8, 11–13, 26–28, 30

SoC System on Chip. 2

SOI Silicon On Insulator. 7, 8, 10

SPENVIS Space Environment Information System. iv, ix, 27–30

SPI Serial Peripheral Interface. 11

SRAM Static Random-Access Memory. 7, 8, 10, 43

STL Software Test Libraries. iii, ix, 2, 3, 15–27, 30–42

TID Total Ionizing Dose. 5, 6, 10, 13, 17, 25, 30, 42

TMR Triple Modular Redundancy. 11, 13, 26

TTMR Time-Triple Modular Redundancy. 12

VLSI Very Large-Scale Integration process, combining millions of MOS transistors in a single chip. 1, 32

WAR Write-After-Read hazard. 21

WAW Write-After-Write hazard. 21

WC Write-Clear register. 23

WO Write-Only register. 23

ZOIX Fault Simulator software from Synopsys. iv, ix, 32, 33

List of Symbols

ρ	Density of SEE's target material	$\frac{mg}{cm^3}$
E	Particle energy	MeV
LET	Linear Energy Transfer	$Mev \cdot cm^2/mg$
x	Particle range	cm

Abstract

Space has always been fascinating to humans, since the dawn of civilization. From the first astronomers and philosophers of ancient times, who looked at the night sky searching for answers, to the scientists and engineers of modern missions, commanding space probes to the edge of the solar system, space has always been at the epicenter of scientific discovery and human curiosity. From the launch of Sputnik 1 in 1957, to the robotic rovers exploring Mars, space missions have always relied on the latest technological advancements in order to enable physical or remote exploration of celestial bodies. Traditionally, designing a space computer required significant amount of resources, leading to designs with impressive radiation performance records. However, such designs were lacking computational performance, required years of development and as a result increased the total cost of the mission.

In recent years, the advent of CubeSats meant that access to space became available to a wider community of enthusiasts, researchers and private companies who were developing low-mass spacecraft made out of Commercial Off-The-Shelf components (COTS). These components however, were designed with the goal of maximizing performance and power, with little to no flight heritage. Several novel technologies were proposed in the field of error detection and mitigation, in an effort to bridge the gap between COTS processors and their radiation-hardened counterparts. Even though the commercial semiconductor industry has increased the reliability of its products by continuously improving their designs and processes, CubeSats or other low-mass spacecraft that use these components are still less reliable than their larger counterparts.

Given the aforementioned, this thesis aimed at exploring the latest developments in the field of space embedded systems and error detection techniques, in an effort to produce a software flow able to detect faults with increased compatibility across processor models. In order to accomplish this goal, the thesis was carried out at ARM Limited, as part of the Software Test Libraries (STL) team responsible for developing efficient assembly tests for detecting random faults. The Cortex-M55 CPU was chosen as the test-bed for this work, in order to develop STL routines for a reference module. The Main Interface Unit (MIU) was chosen to act as the proof-of-concept, since it is an important module in every Cortex-M processor, interfacing the core with the main memory.

More specifically, a series of tests were developed for every major module within the MIU. The design started from the largest module first, which yielded a good trade-off between coverage and time. The tests comprised of efficient assembly routines designed to trigger specific memory access patterns, targeting different portions of logic each time. At the same time, a verification software flow was developed in order to test the newly designed routines against a multitude of possible configurations and initialization parameters. This activity was necessary to ensure that the developed software will be able to operate in a variety of end applications, either in the context of a Real-Time Operating System or baremetal application.

The developed STL tests were subjected to a series of fault simulations using a state-of-the-art hardware simulation tool called ZOIX. Permanent faults caused by accumulated damage were modeled as stuck-at-faults, whereas transient soft-errors were modeled as transient toggle faults. Determining an accurate fault injection interval, required knowledge of the radiation environment that a COTS-based mission would encounter. A state-of-the-art space simulation environment called SPENVIS was used in order to acquire metrics on selected reference missions on Low Earth Orbit and Geosynchronous Equatorial Orbit. This helped setting the upper limits on upset rates, which were in turn used during fault simulation to recreate realistic conditions.

The developed software tests exhibited solid performance in detecting permanent faults, while achieving promising results in transient fault simulations, given certain assumptions. A series of recommendations is given for future research work on the current framework, in an effort to learn from the challenges faced and tackle some of the identified limitations. Given certain assumptions, there is evidence to believe that STLs could be indeed used for random error detection in future CubeSat missions, without increasing the total cost disproportionately.

1 Introduction

This Chapter presents the background information necessary to understand the context of the thesis objective. More specifically, Section 1.1 includes a short summary of the efforts made towards human spaceflight and highlights the challenges faced when designing a space computer. An introduction to the thesis report is given in Section 1.2, whereas Sections 1.3 and 1.4 provide additional information on the company and the team where the graduation project was performed.

1.1. Background information

Space Exploration has always been in the center of scientific interest. From the first astronomers, to the recent exploration missions, space has always been intriguing and mysterious to the human kind. In recent years, Space exploration relied heavily in the development and deployment of uncrewed vehicles, used to physically or remotely explore celestial bodies of scientific interest. The motive for exploration, lies on human curiosity; it is in our nature to explore the unknown, discover new worlds and answer fundamental questions regarding our existence.

Traditionally, space missions required the use of cutting edge technology, across all disciplines involved with the design and manufacturing of a space vehicle. That was particularly relevant for the fields of electronics and computing. With the advent of the Integrated Circuit (IC) and the design of silicon chips, the first electronic computers emerged, with the most notable being the Apollo Guidance Computer (AGC) [97]. Traditionally, the design of a space computer has been considered challenging due to the following correlated factors, namely power consumption, performance and radiation tolerance. This is because improving one or two of the aforementioned factors, always results in damaging the third one.

As a result, engineers have to always balance these factors in order to produce designs that satisfy certain mission requirements, without sacrificing the reliability and hence the success probability of the whole mission. These systems required years of development and testing, which only added to the total cost and time of the whole space program or mission. Historically, designing and manufacturing complex VLSI circuits is very costly and time consuming, therefore for a system to be economically viable, it will have to be massed-produced and satisfy a wide range of target applications.

In the meantime, the industry has successfully showcased the design and deployment of reliable, low-cost and high-performing computers, in the form of embedded systems for safety critical applications. These applications range from automotive safety and braking systems, to industrial applications that need to operate in harsh and noisy environments for extended periods of time. The rise of the CubeSat community and the New Space age of private companies launching spacecraft and other payloads in LEO and beyond, meant that a wider academic, industrial and enthusiast community was building low-cost, low-mass and small-volume spacecrafts made out of Commercial Off-The-Shelf components (COTS). The computing systems powering these spacecrafts were massively produced with the goal of optimizing the two factors mentioned before. However, they were primarily designed for terrestrial applications with little to no flight heritage.

The continuous push to reduce the development time and cost of spacecrafts, has resulted in spaceflight programs that use increasing amounts of COTS and taking advantage of their higher densities, better performance, low-power consumption and decreased lead-times [50]. Subsequently, new efforts have been made by many space agencies to perform testing campaigns in order to characterise different COTS electronic components. The goal of such effort was to determine the most reliable components that could be used in one or more subsystems, reducing the number of radiation-hardened components used, but without compromising the success of the mission. In addition, several error mitigation techniques would be implemented on the system level or in software, to increase the radiation tolerance of such devices.

1.2. Introduction to the thesis

The goal of the project is to research and design an efficient and cost effective method to improve random error detection and mitigation of COTS microprocessors, with increased compatibility across devices. Commercial grade processors are used both in terrestrial and extraterrestrial applications, ranging from automotive to advanced navigation systems on-board small satellites or CubeSats in Low Earth Orbit (LEO). These systems have to operate in harsh environments, sustaining multiple faults due to radiation or environmental noise. Currently, most error detection mechanisms either rely on purely hardware or software solutions, that are in most cases custom-tailored to a specific device or model. This project aims at exploiting the state-of-the-art developments in software and CPU design, in order to prototype a software flow able to detect and mitigate random faults with increased compatibility across CPUs. To that end, the thesis was carried out at ARM, as part of the Software Test Libraries (STL) team responsible for developing software tests that are able to detect random errors.

The report has the following layout. Chapter 2 offers an overview of the current state-of-the-art error mitigation solutions and sets the framework necessary to define the main Research Questions (RQs) and the Research Objective of the thesis. Chapter 3 presents an introduction to the problem statement, along with the detailed objectives of this work, followed by Chapter 4 which elaborates on the steps taken to satisfy the aforementioned objectives. In addition, Chapter 5 presents an analysis on fault modeling and the corresponding radiation environment simulations performed to determine realistic upset rates. Furthermore, Chapter 6 presents the results achieved in the context of this work, while Chapters 7 and 8 elaborate on the conclusions and recommendations respectively.

1.3. ARM Limited: The Company

ARM Limited is a multinational fab-less semiconductor and software company, based in Cambridge UK [3]. It designs CPUs, GPUs, SoCs, multimedia systems, platforms as well as software development tools and compilers.

The company was founded in 1990 as Advanced RISC Machines Ltd., with the goal to further develop the Reduced Instruction Set (RISC) processor design. Given the nature of RISC, ARM processors are best-known for their low-power consumption which makes them ideal for use in embedded devices such as smartphones, tablet computers and wearables, but also in computers used for controlling electro-mechanical applications such as the Engine Control Unit (ECU) of modern vehicles. Recently, ARM processors have been deployed in infrastructure and supercomputer systems, effectively covering the full spectrum of computing applications.

Processors are designed in one of ARM's Instruction Set Architectures (ISA) and licensed to customers wanting to fabricate a chip as a hardware Intellectual Property (IP). The company offers a broad portfolio of CPUs, ranging from low-power embedded processors to server systems. Currently, the majority of ARM's core business relies on licensing CPUs and GPUs and other hardware IP to the embedded computing market, with ARM technologies reaching up to 70% of the global population. In total, the company has shipped more than 130+ billion ARM chips, creating a vast amount of partners [3].

1.4. Software Test Libraries team

The STL team, is part of the CPU-Engineering team and is responsible for the development of STLs, that enhance the functional safety technology of modern CPUs according to ISO 26262 ASIL D [4]. STLs are efficient assembly tests that are executed either on system startup or periodically, in order to stress and test specific parts of the target CPU and identify random errors, that could lead to single-point failures. They have been proposed for use in safety critical embedded systems, however they could be expanded and used in conjunction with other mechanisms to detect radiation-induced errors.

The team works closely with the designers, and has access to the CPU designs from an early stage. Therefore, the team is able to write efficient tests that provide a large coverage with respect to the total number of possible faults. In addition, the team provides feedback to the designers, in cases where there are difficulties in targeting a specific module, or in cases where the coverage results are not adequate. This feedback includes suggestions on how to improve the coverage numbers as well as proposals for additional safety mechanisms, that could be used together with STLs to increase the reliability of the target processor. Finally, the team develops the tests and accompanying verification software flow in such a way, to make it as portable as possible across multiple projects. This in turn reduces the development time of safety tests for future projects, reducing the impact on the time-to-market of a specific design.

2 Research Framework

This Chapter summarizes the work carried out before the beginning of the thesis in order to define the Research Objective. This was accomplished by performing a state-of-the-art review of the current work in the field of embedded space systems and error mitigation mechanisms. Furthermore, this Chapter presents the main conclusions drawn which in turn helped define the purpose statement of the thesis.

2.1. Research Questions

The main goal for such research was to identify potential areas of improvement and make a contribution to the respective body of science. For that reason, the Research Questions were formulated in such a way in order to better understand the nature of radiation, the effects of random faults on modern CMOS circuits and the current methods used by academia and the industry to detect and mitigate them.

To achieve that, one or more of the following questions must be answered:

- **RQ-1:** How does the space environment affects modern semiconductor systems, which are the effects and how these evolve during the operational lifetime of the mission?
  - **RQ-1.1:** What is radiation and how it affects electronic devices during the lifetime of a mission?
  - **RQ-1.2:** What are the types of radiation effects on modern Complementary Metal-Oxide-Semiconductor (CMOS) devices in space applications?
- **RQ-2:** Which are the most prominent methods in the industry, to detect and mitigate errors and which techniques are implemented in widely-used processors for space applications?
  - **RQ-2.1:** How random errors are detected and mitigated in computing systems, in general?
  - **RQ-2.2:** Which error detection and mitigation techniques are implemented in widely-used processors for large spacecraft or CubeSat missions and what is their radiation performance?

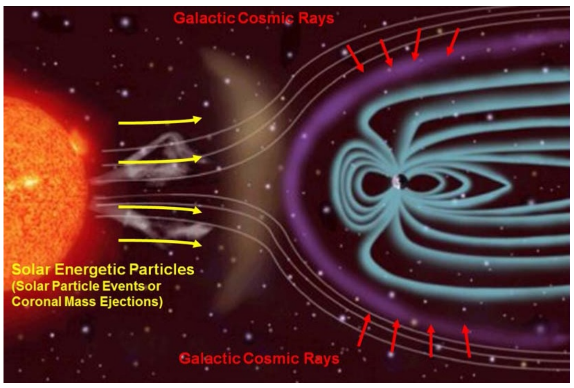
The sources of radiation and its effects on IC devices will be elaborated in Sections 2.2 and 2.3. Section 2.4 will focus on the most prominent type of radiation effects and analyze the underlying mechanisms that govern their behavior. Furthermore, Section 2.5 will present the most prominent error detection schemes used in COTS or radiation-hardened CPUs, whereas Section 2.6 lists the most preferred mitigation methods in a selection of widely-used processors. Finally, Section 2.7 will elaborate on novel detection techniques proposed by scholars or the industry and Section 2.8 will summarize the main findings and conclusions of the said research.

2.2. Sources of radiation

In essence, radiation is the interaction of highly charged particles with matter, resulting in depositing energy into the target object or device. This interaction can be broken down to three underlying mechanisms which can occur with different probabilities [24, 26].

There are three ways a charged particle can deposit energy into an object. First, by interacting with the electrons of the target's atoms, pulling them from their orbits and creating pairs of electrons and holes. This effectively alters the electrical behavior of devices, by changing for example the threshold voltage of a Metal-Oxide-Semiconductor Field-Effect Transistor (MOSFET) [80]. Second, radiation's interaction with objects is also evident when highly charged particles hit dense matter. The loss of kinetic energy creates photons via the Bremstrahlung Effect, which consequently results in ionization traces deep inside dense matter. The third and least frequent way of depositing energy is by interacting with the target's nucleus. In extreme cases, charged particles interact with the nucleus though electromagnetic or nuclear reactions causing displacements or even fractures. As a result, the lattice sustains severe damage especially when this effect is prolonged for extended periods of time.

Radiation has three primary sources. The first source consists of protons and electrons which are trapped by the Earth's magnetic field, forming what is known as the Van Allen Radiation Belts [24, 26]. These particles originate mostly from Cosmic Rays and Solar Flares. In the case of the former, the rays originate from supernova explosions or other events outside the Solar System and consist of protons, electrons and ions [24, 71]. The remnant magnetic field from the supernova, as well as other electromagnetic fields accelerate these particles up to thousands of GeV, effectively converting them to Cosmic Rays [16, 24]. In the case of Solar Flares, they are usually associated with the ejection of plasma and particles from the Sun and are visually perceived as sudden flashes of increased brightness [24, 26]. If directed towards the Earth's vicinity, they are either trapped in the Radiation Belts, or penetrate the atmosphere causing bright auroras. An illustration of Solar Flares and Cosmic Rays is visible in Figure 2.1a, whereas Figure 2.1b provides a visualisation of the Earth's Radiation Belts.



(a) Graphical illustration of Solar Flares and Cosmic Rays [71].

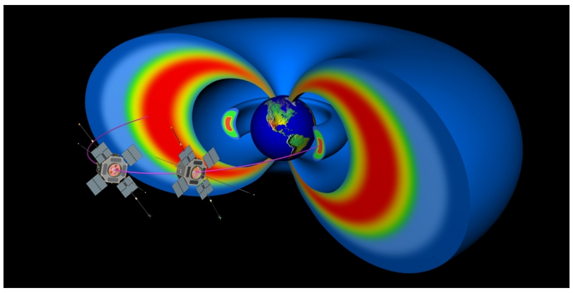

(b) A cross section of the toroidal-shaped Van Allen belts showing satellites near the region of trapped radiation [69].

Figure 2.1: Illustrations of the three radiation sources.

2.3. Classes of radiation effects

Radiation affects electronic devices causing permanent or temporal damage, depending on the incident energy and the location of impact. As a result, microelectronic devices are very sensitive to incoming particles, not only due to the permanent damage that may be caused, but also due to the deposition of charge on sensitive areas. There are three main classes of damage, which will be described below.

- **Total Ionizing Dose (TID):** Total Ionizing Dose is the accumulation of ionizing dose over long periods of time and can lead to the degradation of the target device's lifetime [24, 26]. This cumulative effect alters the transistor's electrical properties, which could continue to operate without any apparent faults. However, the accumulation of ionizing dose will lead to its eventual failure. The most characteristic example of this effect is the accumulation of charge on the gate oxide of MOSFET devices. When highly charged particles interact with the oxide of a target device, electron-hole pairs are generated in the SiO_2 layer. Due to the mobility characteristics [42], as well as a process called "Hopping Transport" charge is accumulated on the gate of the transistor effectively inverting the channel and allowing conduction even at $V_{gs} = 0$, in the case of an NMOS [81]. As a result, conduction in the OFF state not only affects the logic behavior of the device or circuit, but at the same time increases power consumption leading to an eventual burn out. Finally, medium and low dosage rates can cause a deterioration to the timing characteristics of an IC, since the affected electrical capabilities degrade the drive of the MOSFET [81].

- **Displacement Damage (DD):** Displacement Damage refers to the long-term accumulation of non-ionizing dose, due to the interaction of charged particles with the target's lattice [24, 26]. This in turn degrades the electrical behavior of a device and creates background noise [24]. Displacement Damage (DD) damage is usually observed in sensors, amplifiers or Charge-Coupled Devices (CCD).

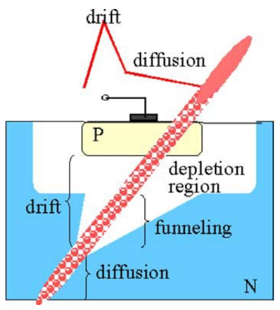
- **Single Event Effects (SEE):** Single Event Effects (SEE) are functional faults on devices and circuit nodes due to the sudden deposition of ionization energy from a single particle, in sensitive regions. Depending on specific conditions, these faults can be either destructive causing short-circuits and burnouts, or non-destructive in the case of functional soft-errors caused by bit upsets and latent voltage swings.

Single Event Effectss are one of the most common radiation effects together with the Total Ionizing Dose and have been extensively studied in the academia and industry. Given their importance, a more detailed examination will be presented in the following Section.

2.4. Single Event Effects (SEE) on CMOS devices

A SEE can be seen as sudden charge deposition on a sensitive node of a device or circuit by an ionizing particle, leading to charge accumulation at the output of a circuit, or where the device connects to. Single Event Effects (SEE) is actually a name used in academia or industry to describe a family of effects comprising of both destructive and non-destructive errors.

An incident ion interacts with the medium's electrons, producing secondary electrons (or δ rays) and therefore electron-hole pairs [24]. This is the same mechanism first introduced in Section 2.2. The radial distribution of charge in the device's matter, is known as the ion track seen in Figure 2.2a. On the other hand, protons do not induce an ion track, but instead interact with matter via elastic or inelastic nuclear interactions with the target device's atoms [24], creating recoil atoms as illustrated in Figure 2.2b. This mechanism is also similar to the general mechanism described in Section 2.2, with the difference being that no permanent damage is induced on the lattice. In order to model the energy deposited in case of a SEE, the concept of Linear Energy Transfer (LET) was conceived, which measures the energy deposited per unit of length into the target device.



(a) Ionization track of an ion hitting the sensitive region of a device.

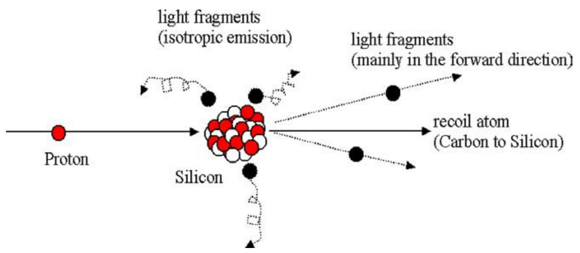

light fragments (isotropic emission)
light fragments (mainly in the forward direction)
recoil atom (Carbon to Silicon)
Proton
Silicon
(b) Inelastic reaction of an incident proton to a silicon atom.

Figure 2.2: SEE mechanisms of interaction with silicon devices from [24].

LET is expressed as follows in Equation (2.1), according to [24]:

$$LET(x) = \frac{1}{\rho}\frac{dE}{dx}(x) \qquad (\frac{MeV \cdot cm^2}{mg}) \tag{2.1}$$

where ρ is the density of target material, E is the particle energy in MeV and x is the particle range in cm. The concept of Linear Energy Transfer can be used in order to calculate the deposited energy into a device or volume of interest and subsequently determine whether an SEE was induced or not. Since the LET of protons is rather low [24], the energy deposition of secondary created particles is used instead to determine the type and criticality of SEE. The different SEE effects are presented in the following Sections, starting from non-destructive effects in Sections 2.4.1 to 2.4.3, followed by destructive effects in Sections 2.4.4 and 2.4.5.

2.4.1. Single Event Upset (SEU)

A Single Event Upset (SEU) is the change of stored state in a memory element inside an IC, due to the sudden deposition of energy from a highly charged particle. The change of state is the result of charge deposition at a sensitive node of a microelectronic circuit, either of a memory cell or register, which subsequently influences the stored value causing a bit-flip or soft-error. These effects are non-destructive in nature, however they can significantly influence the behavior of logic circuits leading to functional failures, system hangs or crashes. SEUs have been one of the largest contributors to device failure both at ground level [23], but also in space [26].

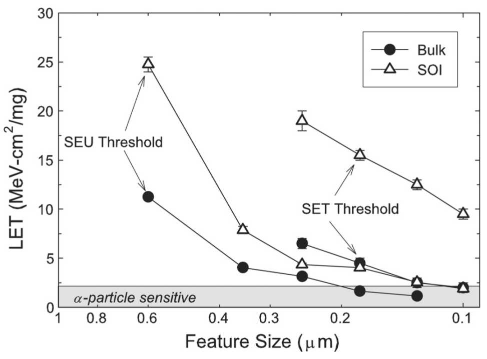
Research in academia and industry has shown that SEUs are observable in areas of a chip that contain a lot of memory elements including, caches, Register Files, Finite State Machines (FSMs), as well as external systems such as Dynamic Random Access Memories (DRAMs). The ongoing research has shown that there is a direct relationship between the amount of charge required to represent information, with the susceptibility to SEUs. More specifically, smaller devices that use lower charge to store information due to technology scaling are far more likely to sustain a SEU [23]. A visualisation of the aforementioned relationship is presented in Figure 2.3a for both Bulk-CMOS and Silicon On Insulator (SOI) processes.

2.4.2. Single Event Transient (SET)

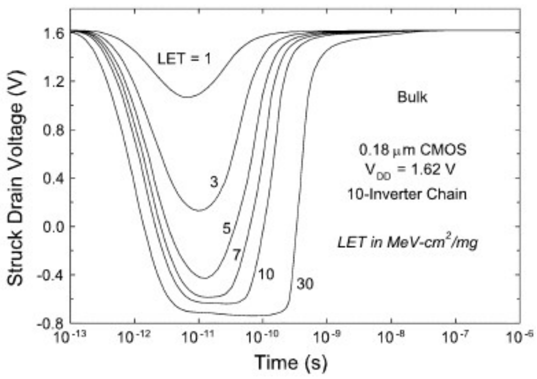
A Single Event Transient (SET) is a temporal voltage or current transient that is generated when charge is deposited from a heavy ion or proton particle passing through a depletion region [22]. The transient is not an error by itself, however depending on the deposited energy and circuit topology, it can propagate through circuit logic and manifest as a SEU on sequential logic or memory arrays [12]. If a SET occurs in the cross-coupled inverter node of a 6T SRAM cell, then depending on the electrical characteristics a SEU could occur. On the other side, if a transient occurs in the combinational logic of a circuit or system, with sufficient amplitude to propagate through the logic and become registered in a subsequent memory element, then the transient may also manifest as a SEU [12]. This of course requires that the transient pulse coincides with a clock edge, otherwise the pulse would not be registered as an upset. As a result, the device susceptibility to SETs depends on the clock frequency and in fact demonstrates a linear behavior [12, 22, 23]. An example of a transient pulse and its relationship with LET is visible in Figure 2.3b.

2.4.3. Single Event Functional Interrupt (SEFI)

A Single Event Functional Interrupt (SEFI) is a non-destructive fault that is observable when a SEU occurs at the control circuitry of a device or subsystem [23]. A SEFI results in the loss of the normal operation, often requiring a reset, power cycle or reconfiguration in order to recover proper functionality. Some typical examples include processor hangs, crashes, or engagement of different operational modes without a command. In the case of Field-Programmable Gate Array (FPGA) chips, the most common errors involve the loss of the device's configuration which requires partial or full reprogramming of the chip [23]. SEFIs are also possible in memory elements such as SRAMs or DRAMs, which contain a lot of control circuitry for address decoding and data multiplexing. A potential strike could cause a series of errors, either fetching data from the wrong address, corrupting the data, or even fetching data without a request. The aforementioned would require a power cycle for full recovery.



(a) SEU LET threshold with respect to device scaling, for SRAMs in CMOS and SOI processes [23].



(b) Simulated transient pulses at the junction of a struck NMOS, in a 10-node inverter chain [22]

Figure 2.3: SEU and SET relationship with device and LET scaling.

2.4.4. Single Event Latch-up (SEL)

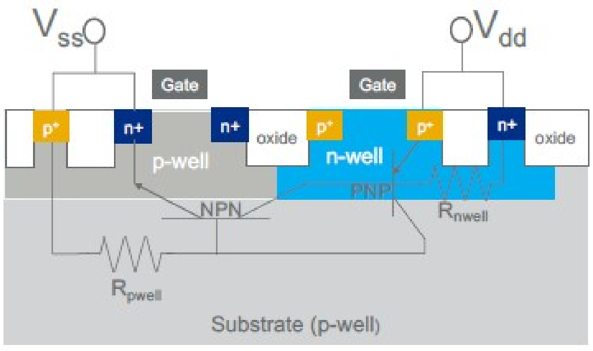
A Single Event Latch-up (SEL) is a destructive Single Event Effect that occurs when a particle strike triggers the parasitic structure inside a CMOS circuit, creating a direct path from the power supply lines to the ground. The parasitic structure visible in Figure 2.4a, is energized when a highly charged particle forward-biases the PNP transistor, which in turn drives the base of the neighbouring NPN transistor formed by the N-well of the PMOS, the p-substrate, the P-well of the NMOS and the n-doped source of the NMOS. This creates a positive feedback loop which increases the current flow until the devices burns out [8]. In that case, power-cycling would be the only option to save the device or IC from a short-circuit. In general, it is difficult to predict SELs, since their occurrence and severity are dependent not only on the radiation environment, but also on the topology of the circuit and more specifically on the distribution of power supply lines and ground nodes in the layout [60].

Researchers have observed that SELs do not necessarily induce destructive damage to the target object, but can trigger a series of latent faults where the device can continue to operate as normal with little to no indication that a SEE ever occurred [10, 60]. More specifically, SEL events can trigger sudden increases in current in the interconnects and metalization layers of an IC, which in turn translate into temperature increases of more than 130 °C [60]. The metal of the interconnect melts, stressing the surrounding insulating materials due to the mismatching thermal expansion [10]. Eventually, this leads to the fracture of the insulating material with the erupting molten metal forming spheres in the immediate vicinity of the fracture. This effect is visible in Figure 2.4b. In certain cases local cooling effects could lead to the crystallization of the erupted metal, forming bridges with the remaining parts of the interconnect. The device would still be able to operate, but with increased interconnect resistance due to the reduction of its cross-section by 1 x to 2 x orders of magnitude [10, 60]. Each subsequent SEL increases the device's susceptibility to latch-ups, leading to its catastrophic failure.

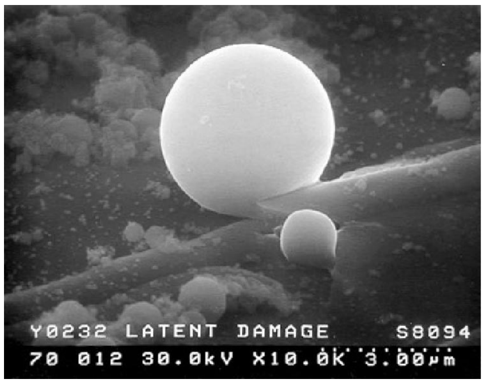
2.4.5. Single Event Burnout (SEB)& Single Event Gate Rupture (SEGR)

A Single Event Burnout (SEB) is a destructive event that is usually observed in Double-diffused Metal-Oxide-Semiconductor (DMOS) transistors used in the power electronics systems of space missions [47]. The effect is similar to a SEL and occurs when a highly charged particle strikes the sensitive region of a parasitic BJT, inherent to the design of every DMOS. More specifically, the parasitic transistor is formed, from the n^+ and n-epi regions, as well as the source n^+ region of the device acting as the collector and emitter terminals respectively. The p-substrate region acts as the base terminal, which if energized by a particle strike as seen in Figure 2.5a, it forward biases the device and turns it ON [47]. The ion track that penetrates the device, creates electron-hole pairs generating a plasma filament that acts as a current-source.The latter drives the BJT, which when turned ON, regeneratively increases the current until the MOSFET fails.

A similar phenomenon is known as Single Event Gate Rupture (SEGR), which also occurs in MOSFET transistors used for power applications. As visible in Figure 2.5b, a particle strike creates an ionizing track in the



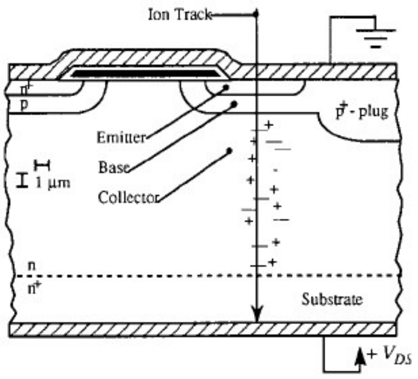
(a) Parasitic bipolar transistors in a CMOS circuit, as illustrated in [8].



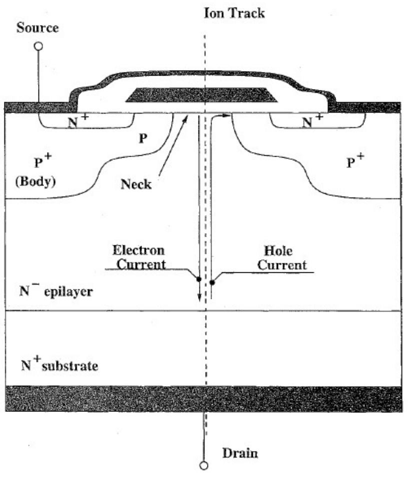
(b) SEM photo of a National ADC10321 device after SEL testing. The erupted Aluminum spheres are visible right next to the fractured and lifted insulator material [10].

Figure 2.4: Examples of SEL damage and underlying mechanism.

silicon substrate. Upon applying a $V_{ds} > 0$, the electron-hole pairs are separated, with the holes moving upwards towards the silicon-oxide interface and the electrons moving downwards to the drain electrode [62, 81]. The presence of holes in the Si/SiO_2 vicinity, induces a corresponding charge in the oxide therefore creating an electric field. Depending on the manufacturing parameters, if the field is increased past a certain value, then the oxide can break causing a Single Event Gate Rupture [62, 81].



(a) A particle strike generating electron-hole pairs, triggering the parasitic BJT device [47].



(b) Ionizing track in a DMOS device, triggering the upward and downward movement of holes and electrons [62].

Figure 2.5: The underlying mechanisms of SEB and SEGR effects.

2.5. Random Error Detection & Mitigation

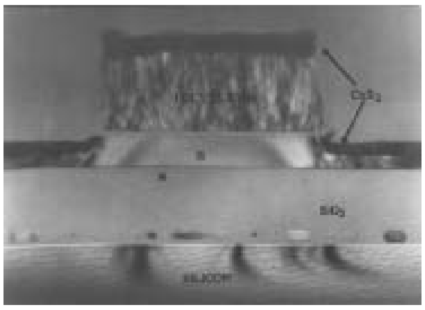
This Section focuses on the most prominent methods developed by academia and the industry in order to detect and mitigate random errors in computer systems. This covers a wide variety of methods ranging from the device and circuit level, all the way to the software and system level. Given the breadth and depth of the field, it is not possible to perform an exhaustive search of all methods in the scope of a thesis project. For that reason, a careful selection was made of the most successful and prominent methods, which were grouped into three broad categories and will be presented below.

2.5.1. Device Level

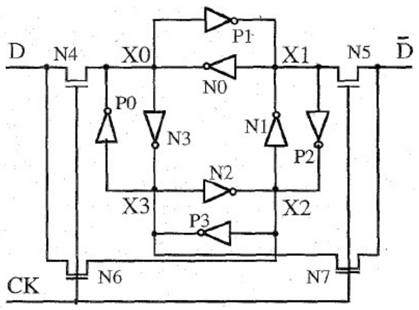
The first category of mitigation methods consists of technologies aimed at the lowest level of the computing abstraction and more specifically at the circuit level. Some methods even extent to the technology or process node, in an effort to further reduce the number of errors and their undesirable effects.

One of the most successful methods in mitigating radiation-induced errors is the Silicon On Insulator (SOI) process. In a SOI process, transistors are built on top of a thin silicon layer that has been deposited on top of a buried SiO_2 layer that acts as insulation [79]. The source and drain regions extent outwards of the aforementioned silicon layer, which is usually hundreds of nanometers thick [81]. The main advantages of the SOI process stem from the enhancements in power consumption and performance. More specifically, the buried insulating layer offers reduced parasitic capacitance with the substrate, reducing power draw and enhancing the ON-OFF characteristics of the devices. However, the most important characteristic that makes SOI the main radiation-hardening technique for more than 20 years is the increased tolerance to SELs [81]. The buried SiO_2 completely isolates the N-well and P-well regions, hence no parasitic BJT structure can exist. In addition, TID effects are also mitigated [20], although the additional trapped charges in the buried oxides interact in complex ways which require further analysis [81]. A cross section of a real SOI transistor is visible in Figure 2.6a.

Other prominent methods involve the design of radiation-hardened libraries including cells with increased tolerances to sudden deposits of energy. These typically include additional elements, as in the case of SRAM cells which may include up to sixteen transistors instead of the standard six. A very successful example of a radiation-hardened cell is the Dual Interlocked storage Cell (DICE) [23], which has been originally presented in [15], but since then has become very popular amongst the community. The Latch implements dual node feedback control using pairs of cross-coupled inverters that store data in complementary pairs of nodes, namely in X0,X1,X2 and X3 as seen in Figure 2.6b. The inverters are actually implemented with either PMOS or NMOS transistors, connected in two opposite loops. The outer clock-wise loop consists of only the PMOS devices, whereas the inner counter clock-wise loop consists of only NMOS structures [15].



(a) Cross section of a transistor implemented in a SOI process. The main layers are visible starting from the silicon substrate, SiO_2, source/drain regions, gate oxide and metal [25].



(b) Principle of operation of a DICE cell, using abstracted inverter symbols [15].

Figure 2.6: Two prominent examples of successful device-level error mitigation.

2.5.2. Microarchitecture Level

This section elaborates on techniques and methods widely used in processors from automotive to space applications, in order to detect and mitigate random faults.

One of the most popular mechanisms that has been widely used in the industry, is the Watchdog Timer [50]. It is a hardware timer that decrements its value every clock cycle and triggers a system reset upon reaching the value of zero [57]. The principle of operation is the following: under normal conditions, a software program will periodically refresh the value stored in the counter, with a process called "kicking" or "feeding" the Watchdog [57], preventing the computer system from resetting. In case there is a software or hardware fault, the computer program would crash and the value of the counter would eventually decrement to zero, thus triggering a reset. Other successful mitigation methods include the use of Error Detection And Correction (EDAC) in the form of parity bits, Cyclic Redundancy Checks (CRC) or Error Correction Codes (ECC) with the most prominent one being the Hamming code [50, 96]. Even though parity and CRC employ only detection schemes and do not correct erroneous data, as in the case of ECC, they have been widely used in memory systems and communication channels in order to detect and re-transmit correct data. Their principle of operation is based on the assumption that it is extremely unlikely for corruption to occur on the same data sequence twice.

A very prominent mitigation technique is called redundancy, which together with the Watchdog Timer and EDAC methods, form a set of standard mitigation strategies used by most modern spacecraft, regardless of their mass and cost [50, 83]. Redundancy relies on the usage of multiple identical hardware modules in order to execute the same operation in parallel, incorporating voting logic at the outputs to detect any inconsistencies or faults in results caused by SEEs. The most common form of redundancy is the Triple Modular Redundancy (TMR), where the hardware modules of a whole logic chain are triplicated, using voting logic that selects the result that two or more modules agree on [50, 83]. Another concept of redundancy involves the triplication of commands executed in one processor, instead of triplicating the hardware, called Time Redundancy [20]. The same command would be executed on the processor three times and the results or checksums would be stored in main memory. A voting mechanism would detect the correct result by selecting the outputs that match. Even though this method offers less system complexity and decreased power draw compared to hardware triplication, it is not as effective in protecting against SEFIs, since a struck bit in the control logic of the ADD function for example, would result most likely in all three copies of the instruction being affected [20].

The concept of hardware and time redundancy can be expanded on the microprocessor level to form fault tolerant computer systems, which perform error detection and correction [18, 46]. Common implementations of such systems involve Dual-Core Lock-Step where two identical CPUs are implemented on the same chip having their own local caches and sharing a common pool of memory. Some implementations enable the execution of the same command with an offset of a few cycles to further increase the confidence in the produced results, since even if a SEU or SET would occur at the same time in both CPUs, it would affect instances of different instructions. Building on the same principle, ARM published a work which used three identical copies of the same ARM Cortex-R5 CPU, with separate clock trees [46]. An error in one of the three processors could be detected and corrected by the matching outputs of the remaining two cores. In case all three outputs differ, then there is no guarantee which is the correct result. However such a scenario has very low probability of occurring in a real system.

2.5.3. System Level

Error mitigation on the system level involves the use of one or more of the aforementioned methods, implemented on the more abstract system level. For example, error detection and correction can apply within the context of a single IC or memory cell, as well as in the context of peripheral systems and buses, correcting almost all SEUs. Many computing systems used in space missions or commercial applications, require the communication between chips in the same or different PCBs, sometimes across different systems. As a result, standards such as the MIL-STD-1773 are used in the aviation, military or space industry in order to implement fault tolerance at the System Level on buses and peripherals [50]. Other methods such as redundancy, are implemented on the system level with the duplication or triplication of the computer systems required for any particular task. For example, the Curiosity rover [38, 99] uses two RAD750 processors, with one serving as the main computer and the other one as a backup [82]. The New Horizons mission [70, 100] also uses two Mongoose-V radiation-hardened processors, with two copies of these systems for redundancy [53].

Other techniques, such as Watchdog Timers and Heartbeat messages can be used on different levels of the computing abstraction to enhance the detection and mitigation capabilities of a given system. For example, Watchdog Timers can substitute for heartbeat messages if the message is sent across devices or system boundaries. If the message is not received within a period of time, then the sender device or system could be considered non-functional [50]. On the other hand Heartbeat messages enhance the status-reporting capability of Watchdog Timers but with the addition of telemetry messages. Large deviations from normal values could indicate a potential upcoming fault and proper action could be taken sooner than later [50]. Heartbeat messages can be exchanged using simple low-level protocols such as I2C or SPI, which are implemented by almost all embedded processors. The messages are still prone to SEUs or other random noise sources, but could be accompanied with an EDAC scheme to boost reliability.

2.6. Mitigation in popular COTS processors

In the context of the thesis, six sample processors were used as a reference point, to research the most prominent methods implemented by the industry in order to mitigate random errors. The sample was taken from both ends of the space computing spectrum, namely the traditional space industry and the emerging CubeSat community. This was done in order to more accurately represent the whole range of available space embedded processors.Three successful COTS microprocessors were used as a reference point for CubeSat designs, together with three prominent radiation-hardened space processors used in many missions, from deep space to planet exploration. The CubeSat microcontrollers were selected upon a NASA study, that was performed for the NASA Electronic Parts and Packaging program (NEPP) [40]. The radiation-hardened counterparts were selected based on their popularity and mission history and include some of the most well known designs used by NASA and ESA.

The three embedded microcontrollers used for CubeSat missions were chosen based on the aforementioned study. These include the Texas Instruments MSP430 series, the Microchip Technologies PIC family of microcontrollers and the SAM9G20 ARM-based line, which was previously offered by the Atmel Corporation. The aforementioned devices were designed and fabricated for usage in embedded applications and can implement one or more mitigation methods as presented in Section 2.5, without using any radiation-hardened components. On the other side, three successful and proven radiation-hardened CPUs were selected from the traditional space industry. The BAE Systems RAD750, introduced in 2000 [56], has a long flight heritage including notable missions such as the Deep Impact spacecraft [19, 51], the Kepler Space Telescope [19, 65], the Curiosity rover [77, 78] and the upcoming Mars 2020 [67, 68]. Furthermore, the Cobham Gaisler GR712RC is a successful implementation of the well-known ESA developed LEON3-FT core, that is scheduled to fly on the JUICE [1] mission, as well as on several Deep Space CubeSat missions [43]. It builds upon a long flight heritage of LEON processors [30], with notable missions such as the ExoMars rover [58]. Finally, the Synopsys Mongoose-V is a popular albeit older radiation-hardened microprocessor used recently in the New Horizons Pluto probe [52]. The technical specifications of all selected CPUs are presented in Tables A.1 and A.2.

After careful examination of the devices' datasheets and a classification of their radiation performance using appropriate academic sources [35, 40, 66], it was concluded that there is a big gap between the radiation performance of COTS processors and their radiation-hardened counterparts as expected, but with a lot of room for improvement. More specifically, almost all embedded CPUs selected in this study featured a Watchdog Timer and some form of EDAC scheme in the internal memories and peripheral buses. Although these mechanisms can be sufficient for terrestrial embedded applications, it is clear that the same can not be said for space. In general COTS devices performed poorly as it is evident from the results presented in [40], with the exception of AT91SAM9G20 which performed very well in terms of latch-up performance. On the other hand, the radiation-hardened CPUs exhibit radiation tolerance or even immunity in SEUs and SELs, given that they implement one or more of the device-level mechanisms presented in Section 2.5.1. A detailed description of the radiation performance numbers for the selected processor models is visible in Table A.3.

From the aforementioned, certain results are very promising and typical of what would be anticipated from radiation-tolerant systems, as in the case of the AT91SAM9G20 device. It is therefore clear that by identifying the gaps in current well-established mitigation techniques and by revisiting some of their strengths or weaknesses, several new opportunities and possibilities would become available for increasing the radiation performance of COTS parts.

2.7. Novel technologies and Reconfigurable Computing

Given the rise of the CubeSat community, there has been an ever increasing need to reduce costs and increase the radiation performance of COTS parts. There has been a series of efforts from companies such as Space Micro which leverages the use of hardware and time redundancy as presented in Section 2.5.2, to come up with novel ideas such as the Time-Triple Modular Redundancy (TTMR) [20]. Other companies such as Vorago Technologies and Ramon.space focus on the device level, by offering radiation tolerant libraries such as HARDSIL [8], or by offering radiation-by-design libraries of custom cells as in the case of RadSafe [55].

Other academic or industrial researchers have shifted their focus towards re-configurable computing, in order to produce efficient open-source solutions for the emerging CubeSat sector. As a result, they implement a lot of the techniques presented in Section 2.5 such as TMR or Time Redundancy, in COTS FPGA chips. Some researchers focus on the development and testing of hybrid techniques as presented in [61, 63], while others have expanded into software redundancy, by exploiting the multi-threading processing potential of modern CPUs and providing software Lock-Step features in a coarse-grain granularity [34]. Furthermore, the advent of re-configurable devices with embedded microprocessors implementing well-established ISAs, such as the Zynq-7000 SoC [102], has the potential to change the way space embedded systems are designed in the near future. By implementing custom hardware EDAC systems in the Programmable Logic (PL) of FPGA chips, as well as taking advantage of the performance of a dedicated CPU, modern system designers have the flexibility to experiment with new heterogeneous computer architectures and potentially provide solutions with improved radiation tolerance.

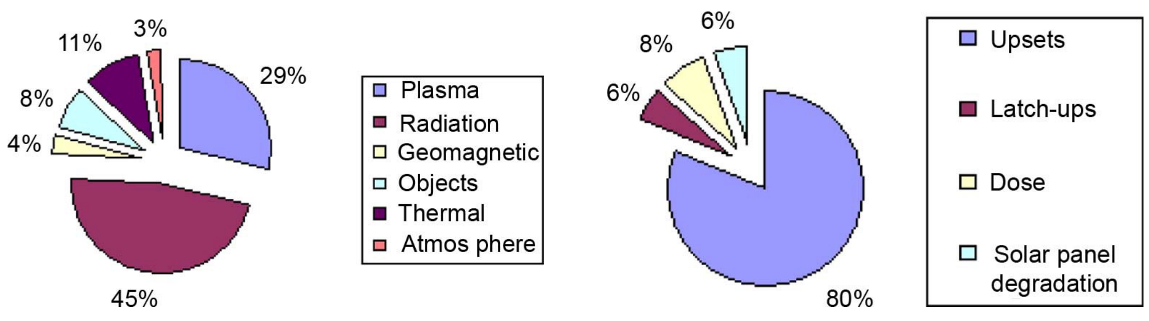
2.8. Conclusions

The research presented above aimed to provide a holistic view on the current status of error detection and mitigation techniques in space embedded systems. From the analysis, it is obvious that mitigating radiation-induced errors is a problem with multiple interrelated aspects. Section 2.8.1 will elaborate on the main conclusions related to the radiation effects on transistors and IC circuits, whereas Section 2.8.2 will focus on current mitigation techniques and their applicability to COTS or radiation-hardened processors.

2.8.1. Radiation effects

Together with the Total Ionizing Dose, Single Event Effects form the most frequent and well-known radiation-induced faults, which have been studied by many scientists and spacecraft designers. In essence, as device size is shrinking, so does their susceptibility to TID, due to the thinner oxide layers associated with reduced sizes. Although, modern CMOS technologies are inherently more resistant to TIDs, the same cannot be said for the SEEs. As presented in Sections 2.4.1 and 2.4.2, shrinking device sizes can have a negative effect on the device's radiation tolerance. As the amount of charge needed to represent information is getting smaller, the susceptibility to SEUs or SETs being manifested as upsets is increasing. In addition, although SELs are mostly dependent on device topology, the smaller cross-sections involved with the reduction in size will definitely have an adverse effect on lath-up performance of future processors and ICs.

A NASA study in 1996 observed that the majority of spacecraft anomalies were attributed to radiation and plasma [11]. From these anomalies, up to 80 % were attributed to SEUs, whereas only 8 % were attributed to TID effects and 6 % to SELs [11]. Even though in the context of that study, the term SEUs includes SETs as well, it is quite evident that non-destructive transient SEEs are the biggest contributor of radiation-induced space anomalies. The results of the aforementioned study are visible in Figure 2.7.



(a) Breakdown of spacecraft anomalies, by their physical cause. (b) Distribution of radiation-induced anomalies.

Figure 2.7: Breakdown of spacecraft anomalies from NASA's study [11], as presented in [26].

The advent of CubeSats using COTS, is only going to make the aforementioned percentage larger. Shrinking device sizes with superior performance to the corresponding radiation-hardened parts, make commercial electronics the Achilles's heel of small-volume, low-mass spacecrafts.

2.8.2. Current mitigation techniques

Almost all of the methods presented in Section 2.5 can be implemented during the design and manufacturing process, with the exception of Time Redundancy and Heartbeat messages which are mostly software related. Therefore, there is some room of flexibility to adapt the behavior of the aforementioned mitigation methods during the lifetime of a mission, up to certain extent. Also, higher level EDAC protocols used could also be adapted or enhanced, assuming that there are no additional hardware requirements. From the previous Sections, it is evident that the vast majority of mitigation mechanisms are implemented in hardware, therefore requiring additional design effort and costs. Furthermore, most mechanisms favor radiation tolerance over performance or power. As a result, engineers and system designers have to constantly balance the aforementioned interrelated factors in order to produce designs that best meet the mission's requirements.

Most of the current COTS CPUs implement one or more error mitigation techniques, however their radiation performance varies a lot. On the contrary, radiation-hardened devices employ many error detection and mitigation strategies across all abstraction levels, at the expense of increased lead times and overall cost. Recently, there has been an ever growing need to bridge the gap between the performance oriented low-cost COTS components and their reliable radiation-hardened counterparts. The latter are not able to bridge this gap, since their superb radiation performance is unfortunately disproportional to their computational performance, power and cost. Several novel solutions have been proposed by the academia and industry in an effort to produce cost-effective and performance oriented solutions, however most of them are still in the early stages of research.

3
Problem Statement

This Chapter focuses on the definition of the thesis objective and topic, by taking into account the results and conclusions presented in Chapter 2. In addition, the main objective will be further divided into sub-goals which are necessary in order to break down the main problem into sizeable and manageable parts. These parts can be latter scheduled to be performed with the correct order or in parallel, making efficient use of the time and resources available. Finally, several restrictions known prior to the beginning of the project will be presented in Section 3.3 providing an overview of the depth and breadth of activities performed during this work.

3.1. Problem Statement

From the academic research performed in Chapter 2, it is evident that only FPGA and software solutions are able to bridge the gap between COTS and radiation-hardened components, without increasing the cost or reducing performance. However, a lot of these methods are the outcome of ongoing academic research, which evolves in dynamic ways. Validating and verifying that the novel hardware/software techniques produce the same results under all circumstances is not a trivial task. In addition, some of the methods presented in Sections 2.5 and 2.7 may be compatible with only one CPU model or design, requiring heavy modifications and re-validation in order to be ported to another design. Other available options either apply externally on fixed COTS devices, or they are applicable during the design and manufacturing phase of few processors with increased prices. Therefore such an approach is not financially viable. Making radiation-tolerant processors available to mass markets, assuming there is a demand for that, would require methods and techniques that are scalable and flexible across multiple CPU models.

3.2. Purpose Statement

Given the concluding remarks of Chapter 2, as well as the ever increasing use of COTS in the space sector, it was decided to explore further in the thesis, ways to improve the error mitigation performance of modern COTS systems. The project would be performed in co-operation with ARM, as part of the STL team responsible for developing STL tests for a variety of processors. STLs are software tests written in assembly in order to stress specific parts of the CPU and identify potential random errors [4]. They have been proposed for usage in safety critical embedded systems, however they could be potentially used in space applications for detecting radiation-induced errors.

The objective of the thesis project was formulated as the following:

"To research and prototype an efficient and cost-effective method to detect and mitigate random faults in Commercial-Off-The-Shelf ARM microprocessors, with increased compatibility across processor models."

More specifically, the main objective can be further divided in the following sub-objectives:

- Familiarize with the ARM architecture and the current infrastructure for Software Test Libraries (STL) development.
- Select a reference CPU model and a target submodule as a proof of concept. Start designing tests with portability in mind.

- Tune performance, to match coverage expectations and iterate as necessary.
- Port all tests to a different processor, within the same architecture extension and verify desired behaviour and coverage.
- Develop a verification flow to prove that each STL routine achieves expected coverage across multiple hardware configurations, without corrupting the system's initial state.
- Provide a unified tool-flow, enabling faster deployment and turnaround time.

Given the wide range of activities needed to satisfy all sub-objectives, it is necessary that one or more tasks are done in parallel. Therefore, even though the list of objectives offers an intuitive methodology with respect to the steps that need to be taken, there have been some alterations in the order and depth, according to intermediate results and scheduling.

3.3. Restrictions

Due to the nature of the problem, as described in Section 3.1 and Chapter 2, as well as the objectives of the thesis, there are a number of restrictions that have been identified during the course of the project.

More specifically, STL development is done in parallel with the design of the target processor. This usually takes several years to complete, before any physical devices are available for testing. Therefore, the work done for this project will capture the development progress of the first 8 months, where the majority of the design and verification milestones are achieved. As a result, it is not possible to test the effectiveness of the tests by performing a physical radiation test, since this would greatly extend over the expected duration of a typical thesis project. Instead, state-off-the-art hardware simulation and fault injection tools will be used in order to introduce random faults on gate-level representations of the reference CPU. These will provide results with enough accuracy to verify whether the tests work and whether the objectives of the project have been met.

In order to realistically simulate the environment that most COTS-based missions would encounter, state-of-the-art space radiation models will be used. These models require years of development and large pools of scientific and mission data in order to be able to accurately predict the radiation environment that a given space mission would encounter. The results of these simulations will be used to further enhance the fault modeling and injection rate used to test the developed prototype flow. This will help reduce the uncertainty gap between the real case and the simulation environment, providing meaningful results.

In general, error injection and simulation is complicated and requires several years and iterations in order to develop tools that have been certified in terms of their accuracy and reproducibility of results. This is especially true for radiation-induced transient faults, whose behaviour is quite complex and their simulation is still the subject of ongoing scientific research. Therefore, even though special care will be taken in order to simulate the aforementioned errors in a certified environment, some deviations and inaccuracies will inevitably be present. To that extent, the absolute numbers in results will not be that relevant, but rather the coverage gains with respect to the baseline.

4
Methodology

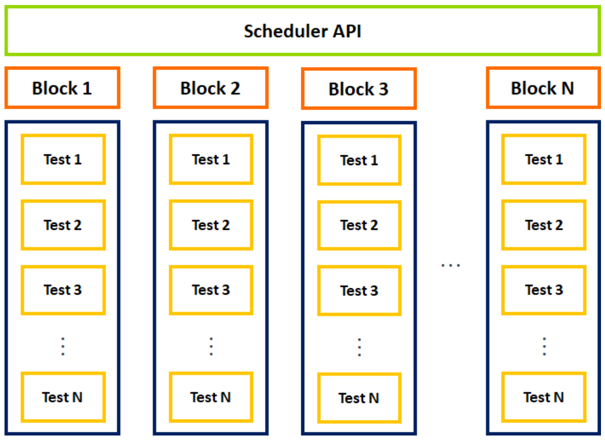
This Chapter focuses on the methodology followed in order to satisfy the thesis objectives as presented in Section 3.2. More specifically, Section 4.1 will elaborate on the STL building concepts and overall architecture, while Section 4.2 will present the reference platform of choice that was used as a test-bed for the current project. In addition, Section 4.3 will present the main design choices and engineering decisions made, while Section 4.4 will focus on the verification efforts aimed at ensuring that the developed software prototypes work under a wide range of system initial parameters.

4.1. Overview: STL architecture

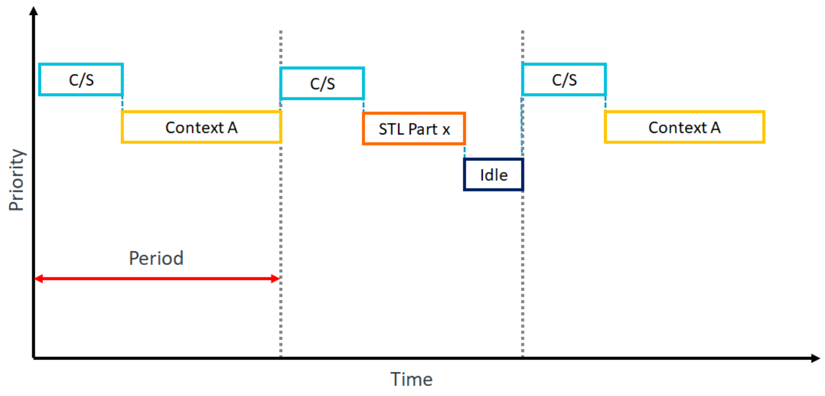
STLs are efficient assembly test routines designed to stress one or more subsystems inside a modern CPU. They play a pivotal role in enhancing functional safety efforts, so that the target systems can achieve ISO 26262 ASIL D level of safety [4]. They can be executed during startup or run periodically with the use of a scheduler software or RTOS, depending on the selected target application. A C-based Application Programming Interface (API) gives the prospective system designers and integrators an easy-to-use and flexible interface to combine multiple STL routines and form larger test suites and programs suited to their needs. STLs are different from a Built-In Self-Test (BIST) in the sense that they are executed periodically, instead of only at startup and do not require any special test or lab equipment in order to be executed. In addition, STLs do not require the CPU to be taken offline while testing, which might lead to reduction in availability. Finally, the context of the processor is not changed, making STLs the preferred method of testing where limited loss of CPU availability is acceptable [54].

The test libraries have been used in the industry to detect stuck-at-faults, which are particularly important in embedded safety applications for the automotive, aerospace and industrial sector. Given their operating principle, there is a possibility to extend them in order to cover radiation faults that have caused permanent damage on the chip, such as the Total Ionizing Dose (TID) and Displacement Damage (DD) effects presented in Section 2.3. Given the fact that STLs are software routines executed at predetermined intervals, it is very challenging or close to impossible to detect latent faults occurring with a higher frequency than the clock frequency, or outside the execution window of the STLs. Hence, fault coverage metrics on SET faults should be expected to be minimal, making the test routines an auxiliary safety measure for these kinds of faults.

A general overview of the STL architecture is visible in Figure 4.1a. Each STL routine is partitioned into one or more blocks, containing one or more parts. In essence, every part corresponds to code targeting a specific submodule or feature within a functional unit of the processor. They are designed in a fully self-contained way and have a bounded execution time of a few thousand clock cycles. As a result, they are deterministic in nature allowing their integration into larger blocks, which in turn are interruptible and relocatable in memory. The prospective user can then schedule one or more blocks of preference, depending on the application requirements of the target computing system. An example of periodic STL scheduling can be seen in Figure 4.1b, where execution is interleaved between the main process and a selected STL part, given a user defined period. In this scheduling scheme, a fixed number of parts is executed per system call and depending on the duration of the part, there might be periods of idle time.



(a) STL software architecture and application interfaces [54].



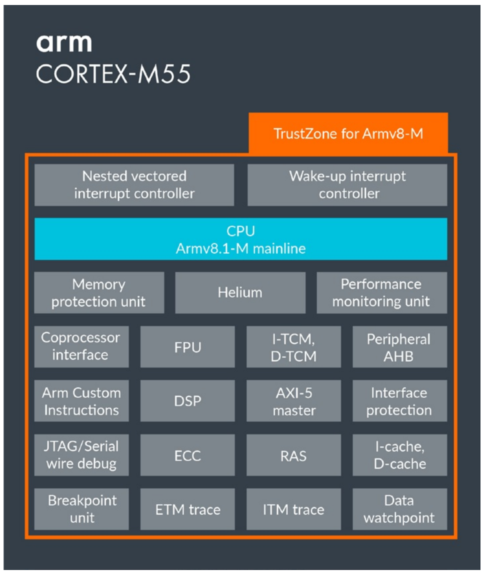
(b) STL execution of a given part, along with the user's application. Context switches are labelled as *C/S*. Adapted from [54].

Figure 4.1: The STL framework, which acts as the starting point of this work.

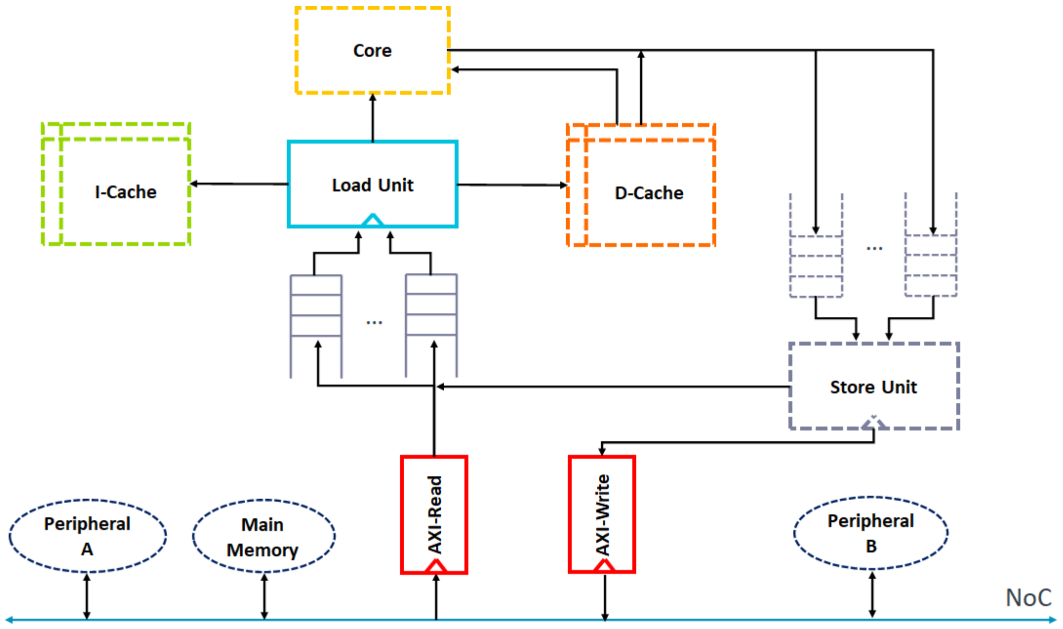
4.2. Reference platform

The aforementioned framework was used as the starting point for this thesis, together with the reference processor. For that, the modern Cortex-M55 was selected, which is a new design implementing the latest ARMv8-M architecture and targeting the next generation of embedded systems, while offering endpoint AI capabilities [5]. This CPU was selected not only for its potential impact on embedded applications, but also for the fact that it is the first M-Class implementing the latest instruction set. This opens up opportunities for reusing the STL routines developed for the project, to future M-class processors, hence satisfying the portability aspects of this work. An overview of the selected platform is visible in Figure 4.2a.

The CPU contains many functional modules, accompanying the main core. These are responsible for the communication with memory, peripherals, or used for debugging purposes and system-level control. In addition, the are many more functional units that have been omitted from the Figure, for the sake of simplicity. STLs can be developed on a per-module basis, starting form the bigger modules and targeting different functionalities. This will help in acquiring higher functional coverage gains, without having to branch out into other modules. STLs have been developed in the past for the main functional units and can therefore be ported to newer designs to some extent, assuming of course that they are modified accordingly. On the other hand, the memory subsystem and more specifically the interfacing units towards the outside of the CPU have been explored less, offering many challenges and rewards. These units play a pivotal role in guaranteeing the correct functionality of any core, since corrupted data arriving in the main execution pipeline will almost certainly generate faults or system hangs.



(a) Block diagram of the Cortex-M55 CPU. Module sizes are not representative of chip area [2].



(b) MIU main components and interfacing modules. The major MIU submodules in terms of size and importance are visible with solid lines.

Figure 4.2: Overview of the reference platform used for STL development during the thesis.

4.2.1. Subsystem selection

Most modern CPUs have an ever growing area of the chip dedicated to system caches, embedded graphics or AI accelerators. Therefore, targeting part of the internal memory system could yield a good trade-off between coverage and time. Given the aforementioned, the Main Interface Unit (MIU) was selected as the target module for STL development, since it is an integral part of the memory subsystem of every Cortex-M processor and has not been exercised yet. It also provides the opportunity to develop processor-agnostic tests, which could be later on ported to other designs. The MIU is also a challenging module to work with, due to the following reasons: it is located far enough from the main core, such that it can only be implicitly targeted with regular load/store assembly commands, while at the same time it exhibits the typical non-deterministic behavior associated with every memory subsystem.

The MIU provides a connection to the outside world, by interfacing with the Network-on-Chip (NoC) using the Advanced eXtensible Interface (AXI), as seen in Figure 4.2b. The latter is a flexible interface capable of parallel high-performance communication on chip [7]. The MIU also routes data to/from neighbouring modules such as the Instruction Cache (I-cache), Data Cache (D-cache) and the main core. Data and instructions arriving via the AXI channel are temporarily stored and routed to one or more appropriately-sized queues. Consequently the Load Unit parses through the pending requests in a FIFO manner, delivering the requested data to the correct consumer module (I-cache, D-cache or core). Data moving out of the core, in the form of store commands, follow a different path towards one or more store queues. A control unit, similar to the one in the read path, parses through the pending store requests and delivers them to the correct recipient. Given the burst nature of the store requests, data are grouped and merged if possible, before being committed again to their destination. The MIU contains the following submodules:

- *AXI Read/Write Registers:* These registers receive and send data over the AXI Interface, using an AXI Master device. The aforementioned device uses five independent transaction channels implemented with FIFO queues; two for Read operations (AR for address, R for data readout) and three for Write operations (AW for address writes, W for writing data and B for write acknowledgments) [7].
- *Load Unit & queues:* The load queues store data in multiples of the word size (32-bit), and use a FIFO scheme in order to deliver data to the controller unit. In turn, this unit arbitrates between multiple pending requests and delivers data and instructions to the appropriate consumer module. In case of data requests, data are always forwarded to the D-cache, meaning that future requests of the same addresses will be directly served from the cache. Finally, the whole load control system can accept and merge store data from the Store Unit, whenever there are pending load requests to the same addresses, or whenever there are pending cacheable stores.
- *Hazarding logic:* This module is responsible for detecting hazards between successive data requests, such as a Read-After-Write hazard (RAW), or a Read-After-Read hazard (RAR) to the same address as the one currently in progress. It is also able to detect hazards associated with cache evictions, since data words may not allocate in the cache until the data they are replacing have been evicted. The module has multiple connections with almost all of the aforementioned modules, therefore it has been omitted from Figure 4.2b for the sake of simplicity.

A regular cacheable load command, results in data being delivered from the AXI Read Register to the load queues. There, depending on the subsequent commands and program flow, data are delivered either to the core, or they are allocated in the D-cache. On the contrary, a regular cacheable store command would follow the path from the core towards the store queues, which collect all store requests. Subsequently, the stores are merged and forwarded to the load path again, in order to be allocated in the cache. Non-cacheable stores or evicted cache data can be delivered directly to the AXI Write Register, in order to be committed to main memory.

In order to effectively stress the MIU, the STL parts will need to be developed in such a way, so that they target one or more of the aforementioned submodules in a meaningful way. In addition, this also helps in breaking the problem of targeting multiple interrelated modules, into manageable parts with defined interfaces and functionality.

4.3. STL Design

This Section elaborates on the design of the STLs, targeting specific submodules inside the MIU. Section 4.3.1 presents the design of the test focusing on the AXI interface, while Sections 4.3.2 and 4.3.3 focus on the load and store paths respectively. Finally Section 4.3.4 elaborates on the hazard detection logic which plays a pivotal role in detecting potential corner-case or harmful scenarios. Each test has been designed to be self contained, using dedicated address ranges, so that they can be used in different scheduling scenarios without the need to run any initialization functions.

4.3.1. AXI Interface part

The first part of the STL design, focuses on the AXI interface. The test consists of three main steps. During the first step, the test generates a series of data, grouped in blocks which are in turn mapped to pre-allocated address ranges in main memory. These blocks contain series of decrementing complementary data, in order to increase the likelihood of detecting a fault in the AXI bus. This is achieved since complementary values increase the toggling of address, data and command lines on the bus, hence offering additional coverage on the AXI module. The aforementioned blocks are committed to memory, during the second step, using the appropriate system control registers as described in the ARMv8-M architecture manual [6].

After the data are successfully written to memory, a series of load commands, reads back the blocks in an interleaving manner. This is the third and final part of the STL, which reads all data values and checks them for consistency. The requests are written in such a way, so that all load queues in Figure 4.2b are equally exercised. Incoming data are checked against their pre-calculated values two times, in order to increase the likelihood of detecting a transient error occurring in the meantime. If all of the aforementioned checks are completed successfully, then this means that there was no fault in the incoming data. Otherwise, any discrepancy detected signals that a stuck-at-fault occurred and as a result, the program branches to a specific fail routine.

4.3.2. Load Unit

The second test focuses on the load path and more specifically on the load queues and control logic which is responsible for delivering instructions and data to the appropriate consumer. This part has a similar structure to the previous one, while differentiating in the way it performs the final load and checking.

More specifically, the load requests have been timed in such a way, as to hit data arriving from the load queues directly, before the Load Unit could allocate them in the D-cache. This helps in increasing the fault coverage of the aforementioned queues, which are an important part of the overall MIU logic. This is achieved through a timed combination of load multiple commands as described in manual for the M-class architecture [6], together with a series of comparison instructions. Incoming data are checked against their pre-calculated values two times- once to validate the data arriving directly from the Load Unit unit and a second time to check the same data arriving from the D-cache. The latter helps in detecting stuck-at-faults that could have occurred in the interface between the Load Unit and the cache.

4.3.3. Store Unit

The Store Unit interfaces with the MIU through dedicated merging paths. It is expected that test routines targeting the aforementioned paths could yield increased coverage results to the Load Unit, since they could directly affect its behavior. In order to accomplish that, a special test case was written in order to stress the feedback loop and its corresponding slot allocation mechanism, which is embedded in the Load Unit. Data patterns are generated, in a similar way to the previous tests, using series of decrementing data values grouped in two blocks. Both blocks are sent to the Store Unit, however a simultaneous read request forces the queues to drain early, re-routing data back to the load path. As a result, the test effectively stresses the merging path between the two modules, hence increasing the potential fault coverage.

4.3.4. Hazarding Logic

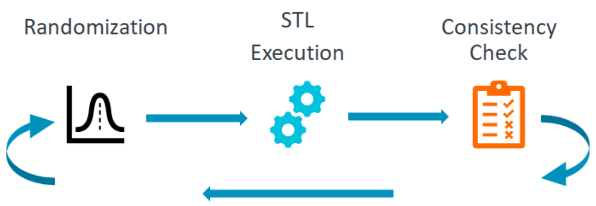
As mentioned in Section 4.2.1, the MIU contains a logic block which is responsible for detecting hazards in the memory subsystem. More specifically, hazards can be created in every pipelined processor, due to the results of one instruction being needed by a subsequent one, before the former is completed [41]. There are several types of hazards such as the Read-After-Write hazard (RAW), the Write-After-Read hazard (WAR) and the Write-After-Write hazard (WAW). Similarly to the pipeline, the same rationale applies to load or store operations sent to the upper layers of the memory abstraction. Modifying data with a series of store commands will generate a data stream, moving from the core to the upper layers of the memory subsystem. Consequently, while the data are on their way through the upper layers of the memory abstraction, a series of load commands to the same addresses will cause a RAW hazard which needs to be detected and serviced appropriately. This is exactly what this test attempts to recreate, by generating blocks of data, storing them and preforming requests on the fly. Attempting to exercise the hazarding logic is crucial for the development of proper STL routines, since it puts the module under test for extreme conditions or corner cases.

4.4. Verification Process

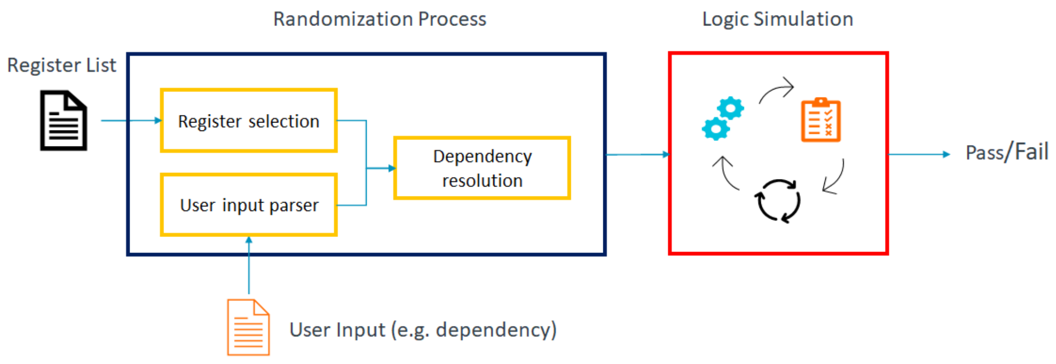
An integral part of designing systems or processes for use in safety systems is the ability to verify that the developed prototype is able to operate under a variety of conditions. In order to achieve that, a testing campaign needs to be devised, which will stress the developed software or hardware module with corner cases, which sometimes might not be obvious even to experienced designers.

As mentioned in Section 1.4, STLs are system libraries developed in order to achieve ASIL D, according to the ISO 26262 Functional Safety standard [4]. These tests can be used by researchers and system integrators in a variety of applications, ranging from the automotive to the industrial or space sector. These applications may require executing software either in a baremetal or RTOS context, with different requirements on the memory footprint, resource allocation and system configuration. As a result, STLs should be able to operate under a variety of circumstances without corrupting the architectural state of the host system, neither its initial system configuration.

In order to achieve this, the designed tests should be exhaustively tested, however the total number of possible test-cases increases exponentially. As a result, there needs to be an automated way of performing testing, proving that STLs can work in a variety of situations. The ARMv8-M Architectural manual lists more than a few hundred system registers, each one having one or more fields that control a specific function or configuration parameter. These registers are accessible to software and can be read or written in order to modify, enable or disable specific functionality in a modern Cortex-M55 processor. A simplified overview of the Verification Process is visible in Figure 4.3a.



(a) Simplified overview of the verification flow.



(b) Randomization and STL scheduling in logic simulations.

Figure 4.3: The STL Verification Process, used in the scope of this work.

The process consists of the following steps:

- *Randomization:* The first step attempts to re-create the many different initial states that a potential system could be set in, while running arbitrary software. More specifically, several possible architectural states are generated by writing randomly constrained values to memory-mapped system registers, which in turn control the processor's configuration and behavior.
- *STL Execution:* Parts are scheduled either in the context of an RTOS or baremetal application, with a variety of scheduling patterns as previously presented in Section 4.1. In any case, part execution should neither corrupt the context of the interrupted client application, neither it should change the configuration of the CPU, by accidentally corrupting one or more system registers.

- *Consistency Check:* The final step reads back the values of the system registers, and compares it with the original ones, prior to STL execution. Any inconsistencies in values, point to a badly written STL part, which corrupts the initial system state.

As mentioned previously, a modern Cortex-M55 core can have more than a few hundred 32-bit system registers, depending on the version and number of implemented features. Given the number of possible configurations, the Verification Process needs to be executed in iterations, in order to cover the vast design exploration space to the best extent possible. These runs are called regressions and are executed in parallel, each one with a different random seed used for register randomization. This way, many errors in STL design can be captured and corrected, that would otherwise remain masked or undetected, when using only logic or fault simulation. As a result, the Verification Process increases the portability and robustness of STL tests, making sure that they will work in a number of different system configurations, targeting a wide range of applications.

In the context of the thesis, it is important to guarantee that the designed STL parts presented in Section 4.3 are properly verified. At the same time, since the Cortex-M55 processor is a new design, there are several modifications required in the current Verification Process. In order to avoid having to rewrite and re-verify the flow each time, it is imperative that the current flow is abstracted and generalized to the extent possible, in order to enhance its portability and enable support for future M-Class processor designs with minimal overhead. This will also help in reducing the number of human errors introduced while performing modifications and reduce the development time. The aforementioned would also satisfy the project's objective of providing a unified and flexible flow, as described in Section 3.2.

4.4.1. Verification Design

One important area of potential improvement is the design of the randomization software, which is the first step in the process, as presented in Figure 4.3b. In previous iterations, randomization was done by manually generating the input register descriptions and hard-coding certain constraints in the Verification Software, so that constrained random values were generated for the respective bitfields. In order to make the procedure as abstract as possible, the whole flow is re-written, using basic parts of the old script and modifying core functionalities such that the end result satisfies the aforementioned objectives. More specifically, the process involves the following steps:

- *Register List:* The input to the flow contains a set of register descriptions for the target processor, delivered by the design team. This list is automatically generated containing standardized descriptions of each register and its respective fields.

- *Randomization software:* This process is an important part of the whole verification flow, as it parses the aforementioned register list and determines which registers to randomize. The ARMv8-M architecture manual defines different types of registers, such as Read/Write (RW), Read-Only (RO), Write-Only (WO) or Write-Clear (WC) registers. The developed software can automatically select which registers can be irritated, as well as which bits to randomize with constrained random values. Additional functionality was added in order to be able to interpret correctly the desired functionality of each bitfield, since sometimes there can be multiple syntax options to refer to the same underlying functionality. Moreover, support was introduced for multiple bit dependencies, as in some cases, certain system registers can only obtain specific values depending on the value of another related register. These dependencies need to be provided in the form of a separate file labeled as "User Input" in Figure 4.3b.

- *Logic Simulation:* After calculating all random values, the simulation step starts, by running the initialization routine. This routine basically writes all pre-calculated random values to all system registers, using dedicated assembly functions. Once initialization is complete, one or more STLs are executed. Depending on the design, the system registers can be checked for consistency either after each STL run, or at the end of the whole simulation. Logic simulation is performed using an industry-standard software provided by Mentor, called Questa [59].

- *Consistency Checks:* After the randomization and test execution steps are completed, the values of all system registers are read back, in order to check whether they have been corrupted during the STL execution. A mismatch in a value of any register, triggers a fault and the process exits with the appropriate error message. Support has been added for WO and WC registers, which read back zero by default as defined in the ARMv8-M manual, in order to avoid false positives.

The aforementioned flow provides the necessary portability required, since it implements a structured way of generating constrained random values for system registers, having a single requirement on the user's dependency files. These files need to be hand-written, only when changing architectures. As a result, two or more implementations of the same instruction set could be accommodated, without the need to change the dependency files. In addition, the list of dependent registers in a given architectural set is rather short. Therefore, creating the aforementioned files by hand is considered a good trade-off when compared to having the whole framework re-adapted or re-written every time from scratch.

4.4.2. Regressions

An important part of the automated flow described in the previous Section, is the ability to launch multiple verification runs that can execute in series or in parallel. Since the number of system registers and potential values is so high, a single run of the aforementioned flow is not enough to uncover all potential errors during the STL design. Hence, by iterating hundreds to thousands of times, many bugs can be uncovered that would otherwise remain masked or hidden.

In general, regression testing is used during software development in order to test the existing part of the code and determine whether the software has regressed after a change. In the context of STLs, the developed code needs to execute correctly in a variety of target configurations, hence testing needs to be performed continuously after every configuration change. This highlights an inherent characteristic of STL verification, which requires a brute-force approach in order to test the developed routines as fully as possible. Another important element of regression testing is early integration, which helps in managing complexity. More specifically, the focus during development was on completing the full Verification Software and creating the aforementioned verification flow, where all submodules were at a satisfactory level of completion, with most of the functionality in place. Consequently, regression support was added in order to start experimenting with the automated flow early on and detect issues with the Verification Design. Later on, once the basic framework was complete, one or more system registers were added incrementally, performing regressions. This helped twofold since the work was split into manageable parts and secondly the whole integrated flow could be tested continuously, along with the introduced randomized registers. If an error occurred, then this would most likely be attributed to the last modification or register added, greatly contributing to the reduction of the required debugging time.

5 Orbit & Fault Simulation

This Chapter will present the fault modeling and radiation environment simulations, that were performed as part of this work. Section 5.1 elaborates on the whole spectrum of faults caused by radiation effects and their modeling in the simulation environment. Section 5.2 focuses on two categories of space missions, where COTS component usage is frequent, making the prototype flow presented in Chapter 4, directly applicable. Finally, Section 5.3 elaborates on the main conclusions of this Chapter.

5.1. Fault Modeling

As seen in Chapter 2, space ionizing radiation has multiple ways of interacting with solid matter and inflicting damage on circuits and devices. This occurs either by accumulating charge over long exposure times, or by sustaining damage in the lattice structure through direct collisions, as explained in Section 2.3. At the same time, radiation also causes functional flaws, by suddenly depositing charge on sensitive nodes of a device or circuit, hence altering its logic behavior.

In order to test the developed prototype STLs and quantify their ability to detect random or radiation-induced errors, it is important to first understand the end effects of faults on logic devices and circuits. This in turn will help to model the faults appropriately, without any loss of generality. Towards that effort, a state-of-the-art fault simulation environment will be used to inject logic faults during hardware simulation. Section 5.1.1 elaborates on the rationale behind permanent fault modeling, whereas Section 5.1.2 will describe the reasoning behind transient fault modeling.

5.1.1. Permanent faults

As already mentioned in Sections 2.3, 2.4.4 and 2.4.5, radiation can cause both permanent and temporal damage [24, 26]. Usually, permanent damage is caused through SEEs such as Single Event Latchups (SELs) and Gate Ruptures (SEGR). A particle impact can trigger the parasitic BJT inherent to CMOS devices, causing regenerative current conduction that can eventually burn the struck transistor. In other cases, SEL-induced peaks in current, causes localized overheating and melting of the interconnect [10, 60]. Even though the device may still be able to function, its susceptibility to future SELs increases, leading to its eventual failure. Latch-up effects can be pronounced in newer CMOS technologies, where the lower metal layers are designed to carry lower currents under normal operation [10].

Another effect of space ionizing radiation which causes permanent faults or affects the timing of sequential circuits is attributed to Total Ionizing Dose (TID), affecting mostly devices that have been subjected to long radiation exposures [75, 81]. This cumulative effect can cause significant changes in the electrical and timing properties of MOSFET transistors, with the most prominent being the conduction of current even in the OFF state [75, 81]. Understanding the underlying mechanisms of TID and DD effects in MOSFETs is quite challenging and several studies in academia have been performed in an effort to quantify a device's response [74]. Even though a device subjected to ionizing radiation effects may exhibit an alteration of its electrical properties at first, it is certain that at some point it will eventually fail [26, 74].

There have been several methods in the industry which allow the detection of permanent faults, such as the BIST, which however reduces the availability of the CPU to the application software and may require additional hardware resources. Furthermore, Memory Built-In Self-Test (MBIST) tests have been proposed, which perform real-time periodic tests of a system's built-in memories. However, they may also require additional hardware resources in the form of dedicated controllers, in order to perform seamless save/restore operations [54]. Finally, Dual-Core Lock-Step is another successful method for detecting permanent faults, as presented in Section 2.5.2. The use of duplicate core resources, is another form of hardware redundancy which provides great fault coverage, at the cost of increased power consumption, chip area and cost. This technique is non-intrusive and transparent to software [54]. However, in all the aforementioned cases, either the processor will be unavailable during a certain time period, or there will be a need for extra hardware resources, which for some applications is not an option. This makes STLs an ideal option for detecting permanent faults, either for terrestrial or extraterrestrial applications.

Modeling in fault simulation

In the aforementioned cases, permanent damage caused by radiation through a series of mechanisms, lead to the occurrence of a fault in the circuit or device. Either by having a transistor conducting without any gate voltage applied, or by burning-out a device or circuit, the logic behavior is permanently affected. The same rationale applies to interconnect damage, where the erupting molten metal either creates voids or shorts with neighbouring metalization layers. The aforementioned faults affect the logical behavior of a gate or circuit, by permanently modifying the logic values at the output of their nodes.

In the scope of this work, permanent faults due to sustained damage will be modeled as stuck-at-1 or stuck-at-0 faults in the simulation environment. This is done in order to accurately map the behavior of such faults to the digital world and more specifically to the inputs and outputs of combinational or sequential logic elements. This fault model provides good gate-level stuck-at-coverage metrics without loss of generality, since any lower level model or mechanism proves either too complex or inaccurate [76]. Logic stuck-at-fault performs well for defects both inside and outside of a cell and can be simulated using state-of-the-art fault injection tools and simulators. This is accomplished by placing faults on input and output terminals, module ports and nets [86]. Some examples of stuck-at-fault placement can be seen in Figure 5.1.

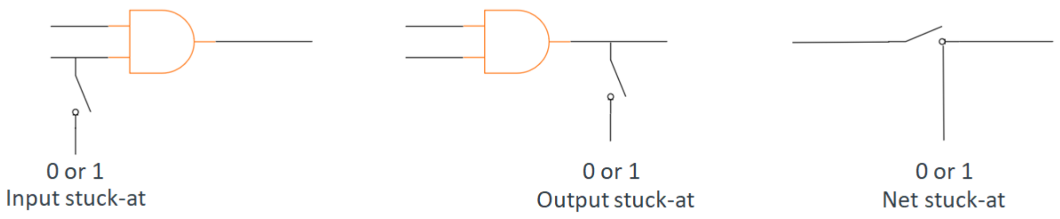


Figure 5.1: Locations of stuck-at-fault placement.

5.1.2. Transient faults

Transient errors are non-destructive SEEs that occur on sensitive nodes of transistors and circuits and can usually manifest in SEUs, SETs and SEFIs. As already seen in Section 2.4.2, SETs are temporal voltage or current transients that depending on the location of occurrence, could propagate and manifest as SEUs in memory elements. On the other hand, SEFIs are functional soft-errors that occur from upsets striking the control logic of a circuit or processor, affecting its normal operation.

As their name suggests, transient faults are faults that occur and then disappear making them extremely difficult to detect and mitigate. They are commonly used in the industry in order to verify designs for automotive or industrial applications, that need to mitigate soft-errors and comply with ISO 26262 and IEC 61508 [86]. Transient faults are becoming ever more important, since they have been reported on ground or high-altitude applications [14, 72]. At the same time, transient faults are also quite relevant for space applications, given the results of the NASA study presented in Section 2.8.1.

Given their importance, there has been a series of mechanisms that were devised in order to detect and mitigate them. The most popular methods involve hardware mechanisms such as ECC logic in memory cells and registers, Dual or Triple-Core Lock-Step and any other form of TMR, as presented in Section 2.5. These methods, have been also implemented in commonly-used COTS or radiation-hardened CPUs, as mentioned in Section 2.6, due to their effectiveness in detecting and correcting transient errors. On the downside however, these mechanisms require extensive design modifications which in turn increase lead times and prices. In addition, they are only applicable during design-time of a specific processor model.

As a result, a more flexible and less design-intrusive software method would be desirable, assuming that it could provide functional coverage from transient faults, but without the inherent disadvantages of the other methods. However, a purely software approach for detecting transient errors, such as STLs, proves very challenging or almost impossible. This is due to the random nature of faults, which can occur at arbitrary times. As a result, if a fault occurs outside the execution window of the test software, then it will not be detected. At the same time, a fault occurring and disappearing faster than the clock period will also remain hidden to the test routine. Hence, there are inherent difficulties when trying to detect transient errors with software, which may limit the applicability of such methods.

Modeling in fault simulation
Given the aforementioned challenges, this work will aim at quantifying whether an STL routine would be able to detect transient faults and to what extent, given a set of conditions and assumptions. The results could be useful in determining whether STLs could be used as an accompanying detection mechanism to existing hardware methods, rather than investigate whether they could replace them.

Faults will be modeled as transient toggle faults, on all possible locations on the target processor module, with a fixed or variable rate. In addition, it is assumed that the transient faults would be occurring during the execution window of the STL routine, since a fault occurring when an STL is not run, would never be detected. Transient faults will be injected in hardware logic simulation as a bit-flip on a target location at specified cycle times, by inverting the value that a non-faulty machine would have at the exact same location and cycle. This type of modeling is called Transient Toggle and is used for both functional safety applications, as well as security applications [86]. Transient faults will be placed on both sequential and combinational primitives, effectively testing against SETs that could propagate and become registered as SEUs.

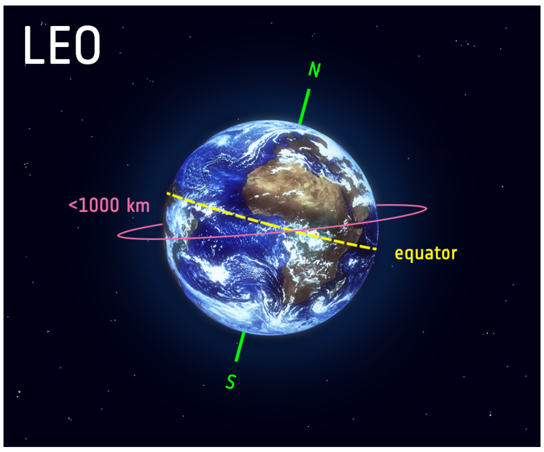
5.2. SPENVIS simulations

The analysis presented in Section 5.1.2 requires knowledge of the circuit topology, as well as the radiation environment, in order to determine where and when to place transient faults during logic simulation. Since the design of the MIU module is given, the number of possible fault locations can be determined. However, the rate at which transient toggle faults occur, not only depends on the the radiation environment, but also on other parameters, such as the position at which the microprocessor or device will be placed inside a spacecraft or satellite.

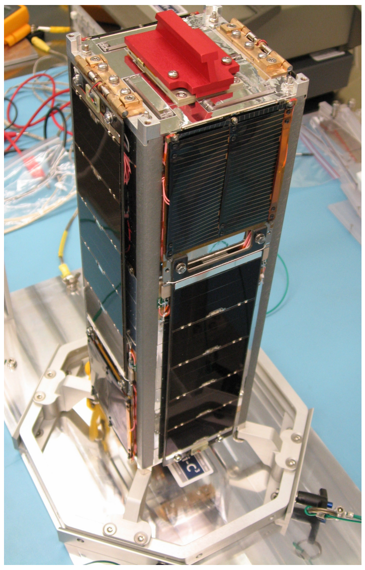
In order to acquire accurate measurements on the error rates, a state-of-the-art space environment simulator will be used, called Space Environment Information System (SPENVIS) which is developed by the Royal Belgian Institute for Space Aeronomy (BIRA-IASB) for ESA's Space Environments and Effects Section [84]. This software platform enables accurate orbit simulation, giving the user the possibility to link the generated orbit to a large collection of space environment modules created by agencies and research institutes around the world. Furthermore, it offers a wide range of models for simulating radiation from the Radiation Belts, the Solar Flares and the Cosmic Rays as presented in Section 2.2. Given the complex and dynamic space environment, it is challenging to acquire results that are applicable to every mission scenario. Hence, it was decided to select two sample type of orbits that were considered the most likely candidates for missions using the prototype software tests developed in the context of this work.

5.2.1. LEO reference missions

Low Earth Orbits are becoming increasingly important for Earth Observation (EO) missions launched by space agencies, universities and private companies. These orbits have relatively low altitudes between 200 km to 1000 km and are primarily used for missions monitoring climate and weather conditions, maritime traffic, the melting of ice in the polar regions, as well as for cartography applications. Sometimes, there is a need to monitor a specific place on the Earth, at exactly the same time of day, as scientists may need to take images of the same place across days, months or even years [28]. Such data can be useful for monitoring forest fires, floods or other weather conditions. A graphical representation of a LEO is seen in Figure 5.2a.



(a) An illustration of LEO, with a reference orbital plane [28].



(b) Final assembly of Delfi-C3 CubeSat [21].

Figure 5.2: LEOs are primarily used for Earth Observation space missions or science missions using CubeSats or similar low-mass, low-volume spacecrafts. A characteristic example of such mission is the Delfi-C3 built from COTS.

A series of private companies have emerged offering accurate Earth Observation data and tools to individuals, such as farmers who would like to monitor their crops. At the same time, many universities and academic institutes have been designing and manufacturing CubeSats, with the goal of launching them to space to acquire scientific EO data, or act as technology demonstrators. A prime example of such an effort is the Delfi-C3 mission launched by TU Delft, which is visible in Figure 5.2b [21]. The aforementioned CubeSat missions are built using COTS components, either for the primary computing module, or for one or more of their payloads.

Two well-known missions were selected from ESA and NASA-CNES in order to be used as a reference for orbit simulations. The first selected mission was Envisat, an Earth Observation space mission satellite providing data for land regions, the atmosphere, as well as the oceans and ice caps [27]. After 10 years of continuous operation, the team at missions control lost contact with the spacecraft, declaring the mission's end on the 9th of May 2012 [27]. The second spacecraft selected for this study is TOPEX/Poseidon, which was a joint mission between space agencies for monitoring Earth's oceans and studying unique ocean phenomena [49]. It remained in operation for 13 years and provided valuable data, such as measurements of sea levels with great accuracy [49]. Both missions were selected not only because they were EO missions from major space agencies, but also because they had an almost circular orbit in close vicinity to Earth, but different inclinations and altitudes. The aforementioned orbits were used in order to get an estimate of the upset rates in LEO, as well as get an upper bound on the number of faults a potential EO mission might encounter, using the Cortex-M55 processor. Along with the two selected missions, a generic one was generated to simulate the environmental conditions at lower altitudes with an inclination of 0°, thus providing additional data for comparison. All simulations were performed using the latest models, for an MIU-sized square module protected with 0.1 cm of Aluminum, acting as the shielding material. The latter was chosen as a conservative estimate, since in most cases the thickness of shielding will be much larger.

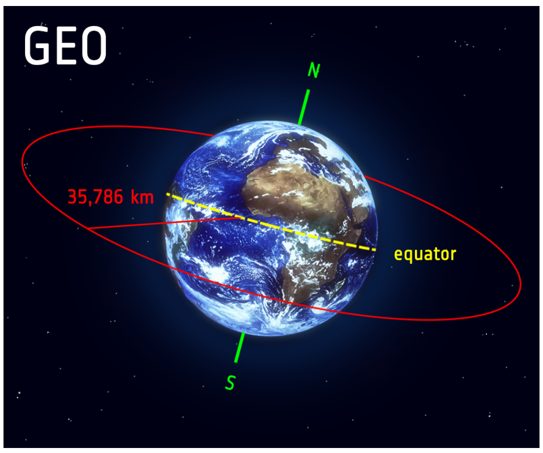
The results of SPENVIS simulations are visible in Table 5.1, reporting the total number of SEU rates for the duration of each mission. The rates calculated can be interpreted as the number of upsets per sensitive node, per unit of time. From the results, it is obvious that a hypothetical MIU-sized hardware module in LEO could experience an increased number of upsets which can reach almost 60 faults per sensitive node per day. Even though missions at lower altitudes can expect significantly less radiation, the aforementioned results highlight the need for effective error detection mechanisms even for COTS processors. A CPU comprises of many modules, therefore the total number of faults would multiply leading to increased error rates.

Specifications	**EVNISAT** [64, 98]	**TOPEX/Poseidon** [48]	**Generic**[†]
Mission Duration [years]	10	13	5
Launch Date	March 1st, 2002	August 10th, 1992	January 1st, 2021
Perigee [km]	772*	1,336	550
Apogee [km]	774*	1,336	550
Eccentricity (e)	0.00042*	0	0
Inclination (i) [°]	98.40*	66	0
Radiation Belts			
Electron model	IRENE-AE9		
Proton model	IRENE-AP9		
Solar Flares			
Flux model	SAPPHIRE peak flux		
Ion range	H to U		
Cosmic Rays			
GRC model $(1\,AU)$	ISO 15390		
Activity data	Solar Minimum (May 1996)		
SEU rate $\left(bit^{-1}\cdot sec^{-1}\right)$	$6.81\cdot10^{-4}$	$5.07\cdot10^{-4}$	$5.00\cdot10^{-9}$
SEU rate $\left(bit^{-1}\cdot day^{-1}\right)$	58.8	43.8	$4.32\cdot10^{-4}$

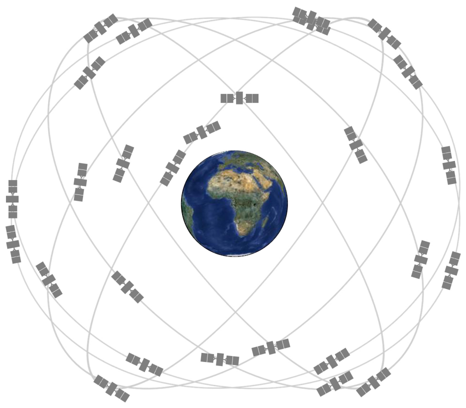
Table 5.1: Simulated radiation upset rates for the selected LEO missions, using SPENVIS. A width of 0.1 cm of shielding was assumed for the electronics compartment. Items with an asterisk $\left(^*\right)$ are best-case estimations based on available orbit data, whereas items with a dagger $\left(^\dagger\right)$ represent a generic orbit to be used for reference.

5.2.2. MEO-GEO reference missions

Medium and Geosynchronous orbits are very common for telecommunication or global navigation missions, such as the Galileo from ESA and the Global Positioning System (GPS) from the US government. They are heavily used by private broadcasters and other telecommunications providers offering internet services, TV or radio coverage to a wide area of the globe. Their main characteristic that makes them so attractive for such missions, is the fact that in GEO an object stays constantly above a certain place over Earth [28]. That way, a ground antenna can be fixed to point always towards the satellite, without the need to move. Typical altitudes for MEO missions are 20 200 km, whereas GEO orbits require a higher altitude at 35 786 km [28]. A typical GEO orbit is visualized in Figure 5.3a.



(a) An illustration of GEO, with a reference orbital plane [28].



(b) Expandable 24-slot arrangement for the GPS constellation, ensuring that at least 4 satellites are visible by any user [73].

Figure 5.3: Communications satellites use GEO staying constantly above a specific point on Earth. Navigation satellites use somewhat lower orbits (MEO) to provide coverage across large parts of the world.

This type of orbits were selected due to the harsh radiation environment that a spacecraft or satellite would encounter at these altitudes. During the early stages of such missions, particles from the Radiation Belts are dominant, whereas Solar Flares and Cosmic Rays become the main contributor to Total Ionizing Dose and upset rates, during the later phases of the mission. In the context of this work, the analysis was performed using the spacecrafts' final orbits, hence more representative results can be acquired by segmenting the mission profile and studying the contribution of every radiation source in each orbit segment separately. Finally, such missions carry telecommunication equipment which should withstand the extreme radiation environment for years. As a result, these orbits present the worst case scenario that modern electronic circuits could encounter upon insertion to their final orbit around Earth.

For the analysis of the radiation environment in MEO-GEO, two well-known and important navigation missions were selected, namely Galileo and the Global Positioning System (GPS). These systems comprise of constellations of satellites, as seen in Figure 5.3b, which are injected into similar or identical orbits over the years to provide increased coverage. They provide services to a wide spectrum of applications, ranging from aircraft navigation to street navigation using a personal handheld device. For Galileo, the GSAT0204 satellite was used for reference, without any loss of generality since all spacecraft in the constellation were injected in similar orbits. As far as the GPS, the latest Block-III satellites were used as the baseline, which as in the previous case, were injected in similar orbits. As in the case of Section 5.2.1, a generic mission was also chosen in order to simulate the environment at higher altitudes and an inclination of 0°. The results are visible in Table 5.2, using the same shielding thickness and reference volume assumptions as before.

Specifications	**Galileo** [17, 29]	**GPS-III** [9, 73]	**Generic**†
Mission Duration [years]	10*	15	10
Launch Date	March 27th, 2015‡	January 1st, 2018*	January 1st, 2021
Altitude [km]	23,222	20,183	35,785
Eccentricity (e)	0	0	0
Inclination (i) [deg]	56	55	0
Radiation Belts			
Electron model	IRENE-AE9		
Proton model	IRENE-AP9		
Solar Flares			
Flux model	SAPPHIRE peak flux		
Ion range	H to U		
Cosmic Rays			
GRC model ($1AU$)	ISO 15390		
Activity data	Solar Minimum (May 1996)		
SEU rate $\left(bit^{-1} \cdot sec^{-1}\right)$	$2.89 \cdot 10^{-3}$	$2.58 \cdot 10^{-3}$	$3.78 \cdot 10^{-3}$
SEU rate $\left(bit^{-1} \cdot day^{-1}\right)$	250	222.8	326.5

Table 5.2: Simulated radiation upset rates for the selected MEO-GEO missions, using SPENVIS. A width of 0.1 cm of shielding was assumed for the electronics compartment. Items with an asterisk $\left(^{*}\right)$ are best-case estimations based on available data, whereas items with a dagger $\left(^{\dagger}\right)$ represent a generic orbit to be used for reference. Finally items with a double dagger $\left(^{\ddagger}\right)$ represent information for the selected GSAT0204 Galileo mission.

As it was expected, higher altitude missions are much more susceptible to SEUs, given their proximity to the Radiation Belts. The average rate increased to almost $3.78 \cdot 10^{-3} \left(bit^{-1} \cdot sec^{-1}\right)$, which corresponds to an impressive $326.5 \left(bit^{-1} \cdot day^{-1}\right)$. This number is extremely high when compared to the LEO missions and as a result, a mechanism such as STLs may not be adequate for handling so many faults, especially given the limitations presented in Section 5.1.2. However, it could be used as an additional line of defence, supplementing an existing hardware method.

5.3. Conclusions

Section 5.1 presented an analysis on the fault models used in logic simulation. Ionizing radiation interacts with CMOS devices in a complex manner, which is still the subject of academic research. In addition, radiation causes both permanent and transient errors on devices, with varying probabilities depending on the topology of the device, the amount of shielding used and the local radiation environment. Hence, performing a logic simulation on the transistor level, injecting both permanent and transient faults is challenging at best and out of the scope of this work. Developing a radiation model for transient faults for example, can be a thesis project or research topic on its own.

Instead, it was decided that faults will be simulated on the gate level, using existing practices and mechanisms in academia and industry. Given certain assumptions presented in Sections 5.1.1 and 5.1.2, radiation fault modeling is possible with stuck-at-fault and transient-toggle logic faults, at specified locations and timings. The aforementioned abstraction is also compatible with the current state-of-the-art fault simulation environments and can produce valuable results. Given the limitations of such tools, as well as the aforementioned abstractions on fault models, it is expected that there will be deviations from real-life tests. Differences in absolute numbers will not be that relevant, but instead the coverage gains with respect to a reference could provide insight as to whether STLs could be used for radiation-induced errors.

Finally, in order to determine the rate at which transient toggle faults could occur, a space environment simulation was performed. Tests on selected orbits, using data from reference missions, provided the upper and lower bounds in terms of upsets per sensitive node and per unit of time. These tests showed that a large number of errors can occur on average per day, highlighting how harsh the space environment can be on electronic devices. However, the error rate for the given number of sensitive nodes in the MIU, results in a fault rate of $\approx 63\ \left(faults \cdot sec^{-1}\right)$. Therefore, the frequency of fault occurrence, is orders of magnitude smaller when compared to the operating frequency of a modern CPU, which is in tens or hundreds of MHz. This is contrary to the original belief expressed in Section 5.1.2, that multiple transient faults would occur in a clock cycle. In addition, it highlights the fact that periodic STL execution could coincide with the occurrence of a transient fault, with a given probability. However, given the calculated low error rate, it is less likely that STL routines could coincide with a transient fault.

6 Results

This Chapter summarizes the achieved results for the prototype flow developed in the context of the thesis. More specifically, Section 6.1 presents the achieved results in terms of fault coverage, whereas Section 6.2 elaborates on the work performed on the verification framework. Finally, Section 6.3 presents some preliminary coverage numbers when porting the developed STLs to a newer Revision of Cortex-M55.

6.1. Fault Coverage

In order to determine the effectiveness of the developed STL routines, a state-of-the-art simulation software was used called ZOIX. This is a proprietary Fault Simulation Environment, that uses a fault injection mechanism to test modern VLSI systems for permanent or transient faults [85, 86]. The tool performs logic simulation of the given hardware design running the payload software, effectively checking for compliance with automotive functional safety standards, such as the ISO 26262, or other international standards such as the IEC 61508.

Given the nature of radiation errors, as well as the analysis presented in Section 5.1, two types of faults were modeled and simulated. The results for permanent fault injection will be presented in Section 6.1.1, whereas transient fault simulation will be elaborated in Section 6.1.2.

6.1.1. Permanent faults

Table 6.1 summarizes the results obtained when introducing stuck-at-faults only in the MIU, using the available gate-level (netlist) representation of Cortex-M55 on the development branch. This provides the most accurate results possible in a simulation environment, since it is using a representation of the design very close to the actual physical implementation of the chip. The fault universe is split into two main categories of faults, namely the Testable and Untestable faults. The former, as their name suggests, consists of all the faults that can be tested in the given design and are taken into account for the final fault coverage calculation. The latter, consist of faults being structurally undetectable, given that they can not be detected by any workload. They are either tied to a supply or ground net (V_{DD} or GND) or being blocked by another signal being tied to them [86]. These faults are not taken into account when calculating the total fault coverage.

ZOIX calculates fault coverage, using the following expression:

$$Coverage(\%) = \frac{D-D+D-F}{D-D+D-F+N-D} \cdot 100\% \tag{6.1}$$

where, Dropped Detected (D-D) and Not Detected (N-D) represent the number of detected and not detected faults respectively, whereas Detected stop/finish (D-F) represents the number of faults that timed-out. The total fault coverage calculated using Equation (6.1) is 32.79%, when taking into account not only the major modules visible in Table 6.1, but also the minor ones which have been omitted for the sake of simplicity.

Given the aforementioned results, it is evident that the current tests achieve good coverage numbers in the largest modules, namely the Load Unit and the AXI Interface module. In the former case, more than 50% of the total faults were detected (DD), whereas in the latter the AXI Interface tests manage to stress 42% of the total interface logic. On the other hand, performance in the other major module inside the MIU, namely the Hazarding logic, is not satisfactory. Even though the tests as described in Section 4.3.4 seem to exercise the

MIU module	**Testable Faults**			Total
	D-D	D-F	N-D	
AXI Interface	12,071 (42.00%)	350 (1.22%)	16,319 (56.78%)	28,740
Load Unit	14,987 (57.27%)	330 (1.26%)	10,853 (41.47%)	26,170
Hazarding logic	475 (4.40%)	191 (1.77%)	10,122 (93.83%)	10,788
Test Coverage	38,816 (31.40%)	1,728 (1.40%)	83,093 (67.21%)	123,637

Table 6.1: Fault simulation results for the major modules only, using ZOIX. Testable faults are listed as Dropped Detected (D-D), Not Detected (N-D) and Detected stop/finish (D-F). Untestable faults are not presented, since they do not contribute to the total fault coverage. A more detailed view of the fault simulation results can be seen in Section 5 of the Detailed Design Report.

subsystem up to a certain extent, it is obvious that the results can be improved substantially, by creating more advanced hazard scenarios. Overall, the total fault coverage of 32.79%, is promising and could be increased further, by revisiting the Hazarding module or exercising the minor modules.

6.1.2. Transient faults

As previously suggested in Section 5.1.2, transient faults will be simulated using the transient toggle model, which introduces faults in all possible nodes at specific times. Since the design has a fixed number of nodes, the only parameter that varies and can influence the results in a meaningfully way is time.

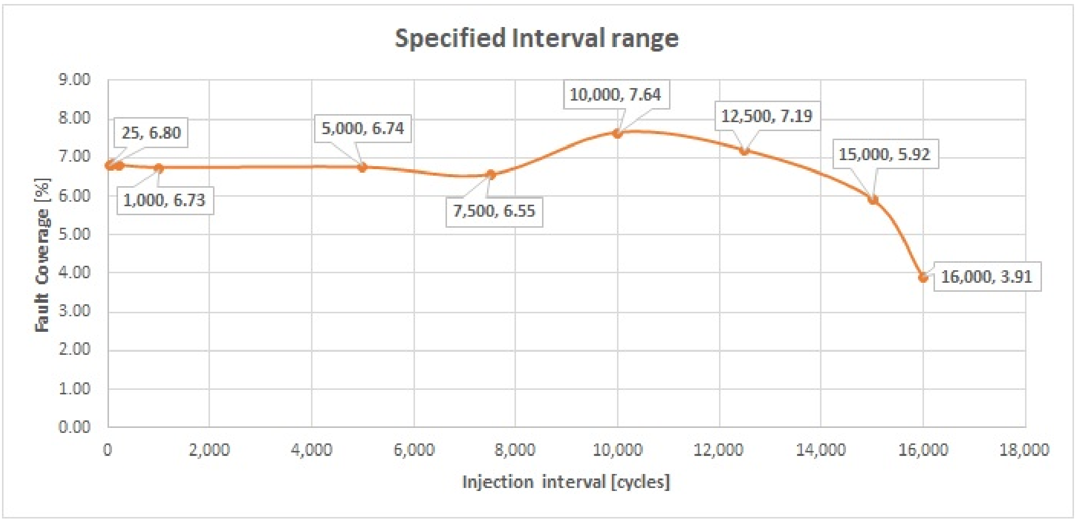
The rates calculated in Tables 5.1 and 5.2 give an indication of the transient error rates that a sensitive device or module such as the MIU could encounter while in LEO or higher orbits. A rate of almost $4 \cdot 10^{-3}$ $\left(bit^{-1} \cdot sec^{-1}\right)$ is possible in the worst-case scenario, averaged for the duration of 10 years in GEO orbit. As elaborated in Section 5.3, the fault rate is orders of magnitude smaller than the clock frequency of the target processor. Hence, it is less likely that a transient fault would coincide with the execution of an STL test.

Determining the probability of periodic STL execution and fault occurrence is out of the scope of this work. This would require extended fault simulations of test execution, together with a RTOS or a baremetal scheduler application. Due to its complexity, such a scenario is not possible with the available computational resources and the current simulation environment.

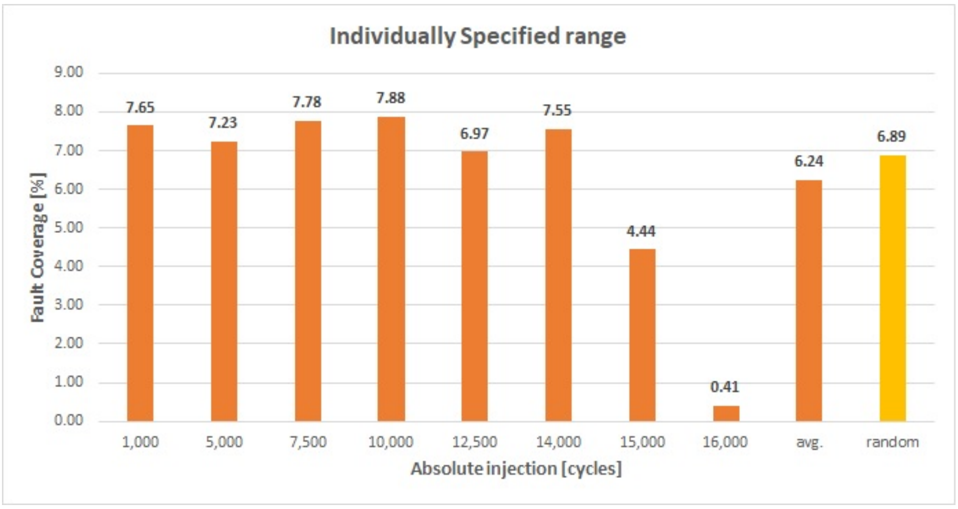
For that reason, this work will focus on the characterization of transient fault coverage, during the execution window of the developed STL tests, assuming that the tests would coincide with one or more transient faults. Given the uncertainty with respect to the time of fault occurrence, an analysis needs to be performed in order to quantify the absolute maximum and minimum coverage possible, in a variety of injection times.

Figure 6.1a presents the results of transient toggle fault injection on the gate-level netlist of the Cortex-M55 CPU, using different toggle intervals. In the context of this fault model, the term interval refers to the duration between two different fault injections. The time scale has been adjusted to multiples of the clock period, for the duration of the whole STL routine. Given the fact that an STL routine completes in the order of microseconds, the analysis is only relevant during the STL execution window. Hence, faults are injected in the given execution window, increasing each time the transient toggle interval for all possible faults in the MIU module. Given the resource-intensive operation of running fault simulations using ZOIX, as well as the number of available licenses for distributed computing, it was not possible to simulate transient faults with an injection interval smaller than 25 cycles. As in the case of permanent errors, fault coverage is reported as the number of detected faults, over the whole fault universe.

From the results, it is evident that fault coverage is significantly lower when compared to the permanent faults analysis in Section 6.1.1. However, this was expected given the inherent difficulties of detecting transient faults with software mechanisms. Overall, for very high injection rates in the order of every 25 to 5000 cycles, fault coverage remains relatively stable above 6.73 %, for the total number of possible faults. From 7500 cycles onwards, the picture changes dramatically. At 10 000 cycle-intervals, the tests reach a maximum of 7.64 % coverage, which starts to drop significantly until the largest interval possible at 16 000 cycles, which is almost equal to the duration of the STL test. At such large intervals, the injection rate becomes lower, having only one or two fault injections and achieving a coverage of 3.91 %. This result is surprisingly valuable, as it highlights



(a) Achieved coverage metrics with respect to increasing toggling intervals.



(b) Fault injection at absolute cycles, along with random fault insertion. Second to last column corresponds to the average of the aforementioned runs.

Figure 6.1: Transient fault simulation results.

a different behavior than the one expected.

As mentioned in Section 5.1.2, faults occurring faster than the clock period could not be detected, since the logic values of all sensitive nodes would be toggled multiple times, before the next rising edge of the clock. As a result, it was expected that the absolute worst-case scenario for fault detection would be at intervals close to the clock period. The results presented in Figure 6.1a portrait a different picture, highlighting that test performance decreases at larger intervals, which correspond to lower injection rates. At the same time, it is observed that coverage is also dependent on time of fault occurrence for a given STL design and can change depending on the toggle activity at the specified time.

In order to quantify the performance of the STL routine at low injection rates, additional tests were performed, where faults were introduced only once at all applicable nodes. The results are visible in Figure 6.1b and injection times are once again reported in cycles. It is evident that the time of occurrence has a big impact on the achieved coverage, ranging from 0.41 % to 7.88 % at 16000 and 10000 cycles respectively. This helps explaining the corresponding peaks and lows in Figure 6.1a. Additional testing was performed using injected faults at random intervals, yielding a relatively high coverage percentage. Overall, the average detection percentage is 6.24 %, suggesting that even though potential drops in coverage are to be expected, on average the STL routine could yield benefits to the deployed system, assuming that its execution coincides with the fault.

6.2. Regression results

An important part of developing STL routines is the Verification Process, as outlined in Section 4.4. The process aims at validating whether the developed software can run on multiple system configurations with a wide range of initialization parameters. Given the number of possible configurations, an automated procedure is needed in order to successfully test as many options as possible, with little to no human intervention.

Hence, the software flow presented in Section 4.4.1 was implemented in order to successfully test the developed routines against a variety of system configurations and identify errors that could potentially remain masked or hidden. The core functionality has been developed in a master program, which in turn generates the appropriate code and subsequently initiates logic simulations. The whole process presented in Figure 4.3 is repeated in iterations generating different random values each time. During the Verification Process, the RTL description of the Cortex-M55 is used, since it is only necessary to verify that system registers have been logically randomized and thus no accurate coverage numbers are required. The result of the verification effort is summarized in Table 6.2, using Questa from Mentor for logic simulations [59].

Verification features	**Status**
System register parsing	✓
Mask generation	✓
Random seed/value generation	✓
Dependency resolution (software only)	✓
RTL Simulation	41/105 registers integrated
Regression support	15 iterations with random seeds

Table 6.2: Status of the verification flow, listing the supported features and functionality. Simulations were performed with Questa Advanced Simulator.

From the aforementioned, it is evident that the end-to-end functionality has been implemented, creating a prototype flow that can be successfully used for regressions. More specifically, system registers supported by the ARMv8-M architecture can be randomized, by generating the appropriate constrained random values for every bitfield. This is achieved by using information from the provided Register List, as well as dependency information which has been manually encoded in the "User Input" file. For every register, appropriately sized masks are generated in order to protect certain bitfields that are marked as *Reserved* by the ARMv8-M architecture, while the remaining fields are randomized. This step is crucial, since writing arbitrary values to *Reserved* fields may have undesirable effects, resulting in system instability, crashes or hangs.

The developed software flow is able to randomize 105/154 registers which have RW attributes, with the remaining registers having hardware dependencies. In order to proceed with the rest of the work and develop an end-to-end verification flow, it was decided not to include hardware dependency support in the current verification software, but instead proceed with the regressions. Having a complete flow working with less functionality was more important than fully developing a specific feature. From the aforementioned registers, a subset of 41 registers has been selected to be integrated in the RTL flow and consequently intro regressions. This also helped in reducing the complexity and breaking the problem of verification into manageable parts. Each register was added incrementally, in an effort to ensure the maximum stability of the developed flow, since a potential crash could be traced back to the last register or parameter added. Multiple successful regressions of 15 iterations each have been performed, in order to validate the developed STLs, as well as the flow itself.

The end result consists of a stable software flow that will be used as the main driver behind the Verification Process of the Cortex-M55 processor. In addition, it proves that the developed STLs work in a variety of initial conditions and system configurations and have been verified to the extent possible. Given the nature of the verification problem, outlined in Section 4.4, as well as the total number of hidden dependencies, it is expected that thousands or even tens of thousands of regressions will need to be performed, in order to fully validate the developed STLs. Hence, the performed work forms the basis of an iterative process which requires significant resources and time to complete, extending beyond the 8-month time frame of the current MSc. thesis.

6.3. Porting to a new Revision

The Cortex-M55 is a new design targeting the next generation of embedded devices and implementing the ARMv8-M Architecture. As mentioned in Section 4.2, the design was chosen based on its potential impact to the next generation of devices and applications, some of which could be targeted towards the space sector. As a result, there are some inherent advantages, as well as challenges, when working on a new and evolving CPU model.

More specifically, the design process of a microprocessor has several phases, each one targeting a specific level of completion of feature implementation, from the total list of functionalities that the new processor would have. Working on a fresh design has many advantages, such as early RTL access in order to start development of STLs as soon as possible, however there are also inherent challenges with constant evolution. Most importantly, the design is under constant change, affecting the functionality and coverage metrics of the developed STLs as well. This is an expected and desired behavior, since incremental evolution means that both hardware and software are getting better over time.

However, this also dictates that the software development should always keep up with the new design changes, adapting to the implemented features and functionalities, in order to match the target coverage. In some cases, design changes may include optimization of existing logic modules, by trimming redundant parts and reducing the size of the respective module. As a result, the total number of possible faults decreases, which may lead to a slight coverage increase. In other cases, logic is further upgraded and expanded with additional features in order to increase the performance of a particular module. In that case, both the functionality and the size of the module are affected, which may lead to an eventual drop in total coverage.

During the development of the STL routines, a specific version of the target CPU was used as the reference platform, which is known as *Revision-0* or simply *r0*. This was necessary in order to have a stable and working system to use as a proof-of-concept. However, given that the processor development was well underway, incremental changes were added frequently, increasing the total MIU module size and affecting its behavior. As a result, when the tests were ported to the latest *r1* Revision of the processor, the aforementioned changes impacted the total coverage, yielding a 20.37 % coverage on permanent faults. It is expected that transient coverage will also be impacted, however due to the time constraints, no fault simulation was performed. Even though the drop is significant, the achieved coverage still highlights the potential of STLs in detecting random faults. It also reveals an inherent limitation of the aforementioned routines, which is their sensitivity to hardware changes. Hence, even though portability can be ensured up to a certain extent, the same can not be said for their effectiveness in detecting faults, which may exhibit drops when porting from one model version to another.

7 Conclusion

This Chapter summarizes the work done in the context of the M.Sc. thesis and elaborates on the achieved outcomes, as well as to what extent they satisfy the original objectives. More specifically, Section 7.1 reinstates the goal of the graduation assignment, whereas Section 7.2 elaborates to what extent the purpose of the thesis has been achieved. Finally, Section 7.3 lists the limitations inherent with this work and discusses the applicability of results.

7.1. Objective statement

The purpose of this work was to develop a prototype software flow that could help in enhancing the radiation performance of COTS processors, in an effort to bridge the gap between their radiation-hardened counterparts. The goal was formulated after a research on the current mechanisms implemented by academia and the industry was performed, followed by an analysis on the effects of radiation on modern CMOS devices and IC circuits. The thesis objective as formulated originally in Section 3.2, is the following:

"To research and prototype an efficient and cost-effective method to detect and mitigate random faults in Commercial-Off-The-Shelf ARM microprocessors, with increased compatibility across processor models."

The aforementioned objective can be broken down into several sub-objectives. More specifically, a representative ARM processor needs to be chosen, in order to act as a proof of concept for this work. In addition, familiarization with the existing framework, methods and practices is required, in an effort to understand how to develop efficient STL routines, on the selected submodule. Finally, tests need to be developed with compatibility in mind and achieve the desired coverage metrics, without corrupting the target system's state or configuration.

7.2. Achieved outcome

Overall, the main objective of the thesis has been achieved, since the prototype software flow was developed for a target ARM microprocessor, acquiring the first coverage metrics on radiation-induced faults for a specific module. This CPU was selected not only for its potential impact on embedded applications, but also for the fact that it implements the latest ARMv8-M instruction set.

More specifically, a coverage of 32.79 % was achieved on a gate-level (netlist) representation of the Cortex-M55 processor, when testing for permanent faults. The current tests achieve good coverage of the major MIU modules, exercising the Load Unit to 57 % and the AXI Interface logic to 42 %. The aforementioned metrics indicate that both modules are stressed to a satisfactory level. Their higher coverage gains, as well as the time spent on designing targeted tests, may hint on reaching a potential point of diminishing returns, where each additional percentage increase would require exponential amount of time. On the other hand, the Hazarding logic test manages to exercise the target submodule, but to a lesser extent, achieving a coverage of 4.4 %. This is expected up to a certain point, since creating complex test scenarios that recreate all possible hazards is challenging. Nevertheless, given the size of the module in terms of total faults, as well as the low achieved coverage, it is assumed that there are more gains to be had from the aforementioned module, that would greatly impact the total coverage of the MIU and in turn the CPU's.

In addition, transient fault simulations were performed, by taking into account the worst-case transient upset rates calculated from the reference missions. The rates were calculated to be in the order of 10^{-3} $\left(bit^{-1} \cdot sec^{-1}\right)$, which for the given set of sensitive bits in the MIU, results in an error frequency of ≈ 63 $\left(faults \cdot sec^{-1}\right)$. This is much lower than the operating frequency of the processor. Given that a complete run of all developed STL parts finishes in several microseconds, there is a probability that transient errors could occur during a periodic execution of STLs. The probability depends on many factors; however given the current framework and its limitations, it was not possible to perform periodic fault simulations. Instead, the thesis focused on determining the maximum and minimum coverage that STLs could yield, assuming that one or more transient faults would occur during the execution window. Therefore several transient fault simulations were run, each time increasing the injection interval, in an effort to quantify performance. It was observed that in all cases coverage remained above almost 4 % and in some cases reached 7.64 %. The latter was not expected and highlighted the potential of STLs being used as a supporting transient error detection technique to the existing methods. In addition, the variation of achieved coverage uncovered another aspect, inherent to the nature of STL routines, that of fault coverage which depends on the time of fault occurrence and varies depending on the toggle activity at the specific time.

Hence, a series of additional tests was performed in order to quantify the fluctuation of results, when injecting single faults only at absolute times. These test cases were more representative of a real-life scenario, given the low upset rates particularly in LEO. It was observed that performance varied from 0.41 % to 7.88 % depending on the time of fault injection, while placing faults at random times yielded a coverage of 6.89 %. Hence, it became evident that the actual worst-case scenario for an STL routine corresponds to times with low injection rates and low toggle activity, contrary to the original belief about high frequency injection rates. On average, the aforementioned tests at specified injection times achieved a coverage of 6.24 %.

Given the aforementioned, it was concluded that STL execution could prove beneficial for the detection of radiation-induced errors, exhibiting solid performance in permanent faults occurring due to the accumulation of ionizing dose or latch-up events. In addition, there is some evidence to believe that STLs could be used as a supplementary detection technique for transient faults, providing up to 7.88 % coverage, on the best-case scenario. Hence, they can be used together with existing hardware techniques, such as ECC in logic, in order to contribute in the successful detection of faults in COTS processors. STLs are a detection-only mechanism, which can provide an early warning to the system integrator or user. Depending on the context, they can be used with traditional hardware methods in order to prevent faults from leading to catastrophic failures. A fault detected during periodic STL execution could mean that the last application software routine might have performed calculations on corrupted data. As a result, the user can potentially re-schedule the execution of the affected software, or perform a system reset effectively mitigating the error.

Finally, the developed tests where integrated into the newly-created verification flow and several regressions were performed, each one consisting of 15 iterations. This helped verifying that the current STLs do not corrupt the initial system state and work as intended, at least on the set of possible configurations tested. Given the nature of the verification work, as well as the whole search space of system configurations, several hundred or thousand runs may be required, in order to fully validate the aforementioned work. However, the exhaustive search of hidden dependencies or software bugs cannot be considered as part of this thesis, which focused on designing a prototype STL and verification flow, as a proof-of-concept.

7.3. Limitations

Given the advantages presented, there are also some inherent challenges identified through the course of this work. More specifically, it has been demonstrated that coverage metrics are sensitive to the underlying hardware changes, which may lead to potential drops when porting from one processor Revision to another. Such was the case when porting the tests designed for *r0* to the *r1* Revision of the CPU. Hence, additional work will be required in order to ensure that the developed test routines yield the same results, across different versions of the same CPU. Furthermore, even though the STLs were developed with portability in mind and are compatible with any M-Class design, it was not possible to port them in another CPU due to time constraints. It is expected that such an activity will require additional time and some modifications on the address ranges used, in order to match the specifications of the new processor. Given the aforementioned sensitivity to hardware changes, it is also expected that coverage will vary. This is due to the fact that the test software was designed with certain stress patterns in mind, which depending on the respective implementation, may or may not exercise the same modules to the desired extent. Therefore portability is to some extent ensured, but definitely requires additional work in order to achieve the same results.

The aforementioned also points to the second major limitation of STL development, which is RTL availability. In order to design efficient assembly tests, it is required that the designer has access to the hardware, in order to understand the cycle-by-cycle behavior and devise test strategies and scenarios that stress large portions of logic. In addition, a hardware implementation is required to test the developed software in logic simulation, as well as access to industry leading tools. This might limit the usage of STLs to some companies or research institutes that have the means to develop their own IP or obtain third-party IP licenses, through collaboration with industry. Otherwise, it would be extremely challenging or close to impossible to acquire a fixed COTS microprocessor and try to develop STLs, while treating the hardware as a black box. This would require additional resources and would take exponentially longer in order to develop tests using a brute-force method. The end result would be inefficient from a coverage perspective and would also require many more than few thousand cycles to complete.

Finally, the acquired transient radiation performance numbers are the result of simulation work assuming that STL execution coincides with fault occurrence. Additional work will need to be performed in order to estimate the probability of transient fault detection, during periodic STL execution. As already mentioned, it is expected that the probability will be rather low, given the space environment simulation results in Chapter 5. In addition, all fault simulations presented in Chapter 6 were performed on a gate-level implementation of the CPU. Even though special care was taken in order to ensure the best possible accuracy, it is expected that there will be deviations between fault simulation performance and an actual physical test in a radiation facility. This is attributed to the fact that radiation effects are complex and dynamic and are still the subject of ongoing research. In reality, both permanent damage through accumulation of dose, as well as transient faults can be occurring with different probabilities at the same time. Hence, developing an accurate model which can be used for fault injection during logic simulations, is a research topic on its own right. The current abstraction models used and assumptions taken, apply to the selected orbit scenarios and are used in this work to evaluate a proof-of-concept. A different model of the radiation environment might result in divergent error rates, affecting the final results. Hence, actual coverage numbers can only be obtained and verified during a radiation test or a potential test flight.

8 Future Work

This Chapter focuses on the strategy and methodology that could be used for future work in this subject or field. The current thesis demonstrated that using STLs for random error detection in COTS processors, can offer increased fault coverage in permanent faults, as well as provide additional detection capabilities for transient faults under best-case conditions. Hence, there is substantial evidence to believe that STLs could be used for radiation-induced error detection in future CubeSat missions, without increasing the total cost disproportionately. Tests can be developed together with custom FPGA implementations of contemporary CPU-IP designs, as part of a license agreement between IP providers and research institutions. However, there are also a lot of possibilities for future additions and enhancements that will be briefly explained below.

8.1. Improving existing framework

More specifically, the current work demonstrated the potential of the STL framework as a random error detection mechanism, however its applicability was proven to a specific module of a selected target microprocessor. Hence in the future, the same analysis could expand to other modules of the same processor, such as the main core or other units within the memory system, yielding even more fault coverage benefits. This will increase not only the total number of permanent detected faults, but also lift the transient fault coverage potentially from single to double digits. Even though, the fault detection performance is not high enough to claim ASIL certification, it is still more than enough to gain the attention of mission designers or system integrators. In addition, portability could be verified by re-using the same STL routines on another microprocessor implementing the same ISA. Coverage will be impacted and additional verification and validation work will be required, however the overhead in time is rather low compared to the advantages offered.

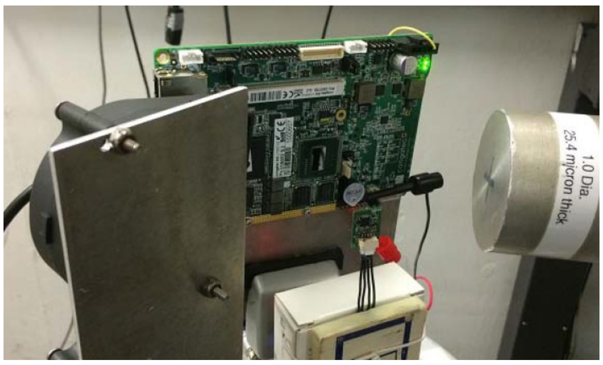
Furthermore, future work could include further iterations to the design of the MIU STLs, in order to bridge the coverage gap between the two processor Revisions, as well as achieve higher detection metrics. Given the current results, it is evident that the Hazarding Logic could be exercised more, by writing more advanced test scenarios that stress a bigger part of the module. This in turn will not only lift the coverage of the MIU module, but also of the whole CPU. The remaining modules could also be exercised further, however it is considered that due to their higher coverage scores, they may have reach a point of diminishing returns. Finally another area of future improvement could be the continuation of the work performed on the Verification Software. By taking the current end-to-end developed framework and expanding it further to support more hardware configurations and even more registers, additional dependencies could be uncovered. This activity will not only benefit the verification design, but also help develop more robust and reliable STL tests, across multiple modules or processor models.

Another area of great importance can be the modeling of radiation-induced faults, in the context of a logic simulation environment. This topic proves to be very interesting and challenging, since creating reliable models and most importantly certifying them can be a substantial endeavor. Therefore, the aforementioned activity could be considered as a future M.Sc. thesis topic on its own right. In addition, there is also the possibility of modifying existing radiation models and integrating them into the current or similar fault simulation environment. As a result, more accurate data could be acquired on the performance of the developed STL routines, proving or disproving certain assumptions and conclusions of the current work. Given the inherent challenges of research and design certification, it is recommended that such activities would be performed with a close collaboration between academic institutions and the industry.

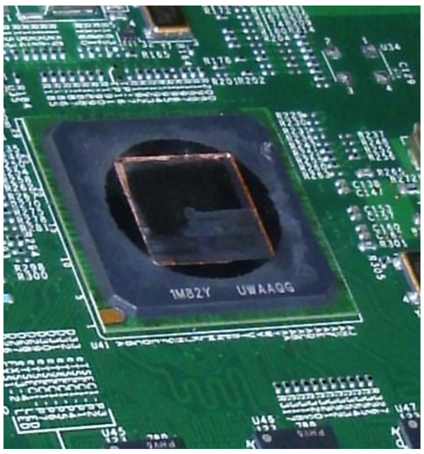
8.2. Physical tests

Another important area of future work involves physical testing of actual devices in an appropriate radiation facility. That way, the target processor device can be subjected to a real radiation environment, reducing the uncertainties and inherent inaccuracies of simulation models. Physical testing for SEEs would require a dedicated test setup and a radiation facility capable of performing high accuracy tests, with capabilities of delivering the required flux of protons or electrons at the desired LET [32].

More specifically, a potential test would require several test chips, as well as testing boards to easily mount the Device Under Test (DUT) and attach the necessary data and power connections, as seen in Figure 8.1a. Mechanical fixtures would be needed, in order to mount the testing board precisely and orient it towards the direction of incoming radiation. Given the dangers associated with radiation, the required test personnel and all monitoring systems should be located in a separate shielded room, to avoid harmful exposures. All test data would be logged in a computer located in the control room. Hence, radiation-induced faults that would lead to data corruption or other functional interrupts could be avoided. Finally, an important step in the process is the removal or thinning of any material in the chip's packaging, which could interfere with the penetration of incident radiation to the sensitive regions of the chip [32]. This is mandatory, since there are not many facilities which could accelerate particles to the same energy levels, as the ones encountered in LEO or GEO. An example of chip preparation for radiation testing is visible in Figure 8.1b.



(a) A Conga QA6 - Intel E620 testing board at a radiation facility [40].



(b) A Freescale P2020 processor, upon removing the packaging layers, to expose the underlying cores [39].

Figure 8.1: Examples of radiation testing efforts on COTS processors.

During the course of the test, knowledge of the chip layout would be mandatory, in order to target specific modules each time. The testing software can include a combination of periodic STL execution, along with a generic payload software to simulate as accurately as possible real-life computing workloads. Each time, a different module can be targeted for transient fault injection, while executing the respective STL routines specifically designed for that module. That way, realistic coverage measurements can be acquired for every single module. STLs could use the existing software infrastructure to report errors to the scheduling program, which in turn would broadcast data through the dedicated cables, to the control room. In total, three chips would be required in order to successfully test a specific processor design and eliminate any lot-to-lot variations, associated with the manufacturing of complex semiconductor devices [32].

Since radiation testing requires chip availability, which might involve additional costs and impose delays in the testing schedule, FPGAs could be used instead. More specifically, in order to begin radiation testing early in the design process, the CPU could be ported to an FPGA, executing the aforementioned STL tests as the payload. This solution could provide results faster, since it does not require any physical design or implementation of the selected CPU model, but rather a synthesizable RTL description. This solution could provide useful insight into the performance of the developed test, but would introduce additional inaccuracies due to the the potential differences in the implementation of the FPGA chip. Furthermore, porting a CPU design to an FPGA is not a trivial task, especially in terms of performance modeling.

Finally, a similar methodology can be followed for TID testing, though the DUT is not necessary to be executing test software during irradiation. The effects of ionizing doze can be evaluated after each test, since it is recommended to perform irradiation at multiple exposures [33]. The device could then be subjected to a series of electrical tests to determine whether there has been any significant device failure. In addition, the DUT could be tested using a series of software tests, by reading or writing memory locations and executing one or more STL tests. That way, the radiation performance of a STL routine could be fully characterized under real-life conditions.

A
Appendix

This Appendix includes the technical characteristics of all selected processors introduced in Section 2.6.

Specifications	**MSP430F161x** [45]	**PIC24FJ256GA110** [95]	**AT91SAM9G20** [94]
Architecture	16-bit, RISC Variant	16-bit, Modified Harvard	32-bit ARM, 16-bit Thumb
CPU (@ max. MHz)	1x MSP430 @ 8MHz	1x PIC24F @ 32MHz	1x ARM926EJ-S @ 400 MHz
Internal Memory (Flash, SRAM)	<55 KB†, <10 KB†	256 KB, 16 KB	64 KB, 2x 16 KB
External Memory (DRAM, SRAM/Flash)	N/A	N/A	≈256 MB*, ≈768 MB*
Unit Cost	15.21 $ [44]	5.29 € [93]	9.16 € [92]

Table A.1: Overview of the most popular microprocessors, for the selected CubeSat missions [40]. Entries with a (*) are best-case estimates based on memory mapping, whereas entries with a (†) are model dependent.

Specifications	**RAD750** [90, 91]	**GR712RC** [37]	**Mongoose-V** [87, 89]
Architecture	32-bit, PowerPC 750	32-bit, SPARC V8 V8	32-bit MIPS
CPU (@ max. MHz)	1x RAD750 @ 200 MHz†	2x LEON3-FT @ 100 MHz	1x R3000 @ 15 MHz†
Internal Memory (Flash, SRAM)	N/A	N/A, ≈197 KB	N/A
External Memory (DRAM, SRAM/Flash)	≈1 GB†, ≈512 MB†	≈1.07 GB, ≈33.5 MB/≈33.5 MB	256 MB*, 192 MB*
Unit Cost	200.000 $ (ref. 2002) [101]	upon request [36]	42.225 $ per 2 units[88]

Table A.2: Overview of the most popular radiation-hardened microprocessors. Entries with a (*) are best-case estimates based on internal memory mapping, whereas entries with a (†) depend on model version.

Processor	TID_{th} $[krad(Si)]$	SEL_{th} $\left[\frac{MeV \cdot cm^2}{mg}\right]$	SEU_{th} $\left[\frac{MeV \cdot cm^2}{mg}\right]$
MSP430F161x [40]	>20	<10	20
PIC24FJ256GA110 [40]	<20	<2	<2
AT91SAM9G20 [40]	-	>85.4	<1
RAD750 [13, 56, 90]	200† - 1000†	>120	>45
GR712RC [37]	<300	>118	Tolerant‡ (>37 ref. FPGA)
Mongoose-V [87]	-	Immune (>100‡)	>80

Table A.3: Radiation performance of all selected processors. Entries with a (†) depend on model version, whereas dash lines denote no available data. A (‡) indicates classification based on information from [35, 66].

Bibliography

[1] Martin Aberg, Daniel Hellstrom, Arne Samuelsson, and Felice Torelli. A Common DPU Platform For ESA JUICE Mission Instruments. *Data Systems In Aerospace Conference Proceedings*, 2016.

[2] ARM. Arm Cortex-M55 Processor. *Product Brief*, 2020.

[3] ARM Limited. Architecting a Smarter World - ARM. `https://www.arm.com/`, . [Online; accessed September 2020].

[4] ARM Limited. Development tools and software: Software Test Libraries. `https://www.arm.com/products/development-tools/embedded-and-software/software-test-libraries`, . [Online; accessed September 2020].

[5] ARM Limited. Central Processing Unit (CPU) . `https://www.arm.com/products/silicon-ip-cpu/cortex-m/cortex-m55`, . [Online; accessed September 2020].

[6] ARM Limited. ARMv8-M Architecture. *Reference Manual*, 2017.

[7] ARM Limited. AMBA AXI and ACE. *Protocol Specification*, 2019.

[8] R. Bannatyne, D. Gifford, K. Klein, and C. Merritt. High temperature/radiation hardened capable ARM Cortex-M0 microcontrollers. *VORAGO Technologies Whitepaper*, Online; accessed September 2020.

[9] Los Angeles Air Force Base. GPS III. `https://www.losangeles.af.mil/About-Us/Fact-Sheets/Article/343728/gps-iii/`. [Online; accessed September 2020].

[10] Heidi N. Becker, Tetsuo F. Miyahira, and Allan H. Johnston. Latent Damage in CMOS Devices From Single-Event Latchup. *IEEE Transactions On Nuclear Science*, 2002.

[11] K.L. Bedingfield, R.D. Leach, and M.B. Alexander. Spacecraft System Failures and Anomalies Attributed to the Natural Space Environment. *NASA Reference Publication 1390*, 1996.

[12] J.M. Benedetto, P.H. Eaton, D.G. Mavis, M. Gadlage, and T. Turflinger. Digital Single Event Transient Trends With Technology Node Scaling. *IEEE Transactions On Nuclear Science*, 2006.

[13] Richard W. Berger, Devin Bayles, Ronald Brown, Scott Doyle, Abbas Kazemzadeh, Ken Knowles, David Moser, John Rodgers, Brian Saari, and Dan Stanley. The RAD750 - A Radiation Hardened PowerPC Processor for High Performance Spaceborne Applications. *IEEE Aerospace Conference Proceedings*, 2001.

[14] D. Binder, E.C. Smith, and A.B. Holman. Satellite Anomalies From Galactic Cosmic Rays. *IEEE Transactions On Nuclear Science*, 1975.

[15] T. Calin, M. Nicolaidis, and R. Velazco. Upset Hardened Memory Design for Submicron CMOS Technology. *IEEE Transactions On Nuclear Science*, 1996.

[16] NASA Goddard Space Flight Center. Imagine the Universe! `https://web.archive.org/web/20121028154200/http://imagine.gsfc.nasa.gov/docs/science/know_l1/cosmic_rays.html`. [Online; accessed September 2020].

[17] European GNSS Service Centre. Orbital and Technical Parameters. `http://www.esa.int/Applications/Navigation/Galileo/Galileo_a_constellation_of_navigation_satellites`. [Online; accessed September 2020].

[18] ARM Community. Evolving safety systems: Comparing Lock-Step, redundant execution and Split-Lock technologies. `https://community.arm.com/developer/ip-products/system/b/embedded-blog/posts/comparing-lock-step-redundant-execution-versus-split-lock-technologies`. [Online; accessed September 2020].

[19] Ball Aerospace & Technologies Corporation. RAD750. https://web.archive.org/web/20070711152307/http://www.ballaerospace.com/page.jsp?page=96. [Archived Online; accessed September 2020].

[20] David Czajkowski and Merrick McCartha. Ultra Low-Power Space Computer Leveraging Embedded SEU Mitigation. *IEEE Aerospace Conference Proceedings*, 2003.

[21] TU Delft. Delfi-C3. https://www.tudelft.nl/lr/delfi-space/delfi-c3/. [Online; accessed September 2020].

[22] Paul E. Dodd, Marty R. Shaneyfelt, James A. Felix, and James R. Schwank. Production and Propagation of Single-Event Transients in High-Speed Digital Logic ICs. *IEEE Transactions On Nuclear Science*, 2004.

[23] P.E. Dodd, M.R. Shaneyfelt, J.R. Schwank, and J.A. Felix. Current and Future Challenges in Radiation Effects on CMOS Electronics. *IEEE Transactions On Nuclear Science*, 2010.

[24] Sophie Duzellier. Radiation effects on electronic devices in space. *Aerospace Science and Technology*, 2004.

[25] D.J. Eaglesham. 0.18μm CMOS and beyond. *IEEE Design Automation Conference Proceedings*, 1999.

[26] R. Ecoffet. Overview of In-Orbit Radiation Induced Spacecraft Anomalies. *IEEE Transactions On Nuclear Science*, 2013.

[27] ESA. Envisat. https://www.esa.int/Enabling_Support/Operations/Envisat, . [Online; accessed September 2020].

[28] ESA. Types of orbits. https://www.esa.int/Enabling_Support/Space_Transportation/Types_of_orbits, . [Online; accessed September 2020].

[29] ESA. Galileo: a constellation of navigation satellites. http://www.esa.int/Applications/Navigation/Galileo/Galileo_a_constellation_of_navigation_satellites, . [Online; accessed September 2020].

[30] ESA. Microprocessors. https://www.esa.int/Enabling_Support/Space_Engineering_Technology/Onboard_Computer_and_Data_Handling/Microprocessors, . [Online; accessed September 2020].

[31] ESA. RACE double CubeSat mission. http://www.esa.int/ESA_Multimedia/Images/2019/06/RACE_double_CubeSat_mission, . [Online; accessed September 2020].

[32] ESCC. Single Event Effects Test Method And Guidelines. *ESCC Basic Specification No. 25100*, 2014.

[33] ESCC. Total Dose Steady-State Irradiation Test Method. *ESCC Basic Specification No. 22900*, 2016.

[34] Christian M. Fuchs, Nadia M. Murillo, Aske Plaat, Erik van der Kouwe, and Peng Wang. Towards Affordable Fault-Tolerant Nanosatellite Computing with Commodity Hardware. *IEEE 27th Asian Test Symposium*, 2018.

[35] Gianluca Furano and Alessandra Menicucci. Roadmap for On-Board Processing and Data Handling Systems in Space. *Dependable Multicore Architectures at Nanoscale*, 2018.

[36] Cobham Gaisler. Product Ordering. https://www.gaisler.com/index.php/information/ordering. [Online; accessed September 2020].

[37] Cobham Gaisler. GR712RC Dual-Core LEON3-FT SPARC V8 Processor. *Device Datasheet*, Revised 2018.

[38] John P. Grotzinger and et al. Mars science laboratory mission and science investigation. 2012.

[39] Steven M. Guertin. SOC SEE (Radiation) GuidelineUpdate. *NEPP Electronic Technology Workshop*, 2014.

[40] Steven M. Guertin, Mehran Amrbar, and Sergeh Vartanian. Radiation Test Results for Common CubeSat Microcontrollers and Microprocessors. *IEEE Radiation Effects Data Workshop*, 2015.

[41] David Money Harris and Sarah L. Harris. Digital Design and Computer Architecture. *Morgan Kaufmann Publishers Inc*, 2017.

[42] R.C. Hughes. Charge Carrier Transport Phenomena in Amorphous SiO_2: Direct Measurement of the Drift Mobility and Lifetime. *Physical Review Letters*, 1973.

[43] Travis Imken, Julie Castillo-Rogez, Yutao He, John Baker, and Anne Marinan. CubeSat Flight System Development for Enabling Deep Space Science. *IEEE Aerospace Conference*, 2017.

[44] Texas Instruments. MSP430F1611: 16-bit Ultra-Low-Power MCU. `http://www.ti.com/product/MSP430F1611?keyMatch=MSP430F1611&tisearch=Search-EN-everything&usecase=part-number`. [Online; accessed September 2020].

[45] Texas Instruments. MSP430F15x, MSP430F16x, MSP430F161x Mixed Signal Microcontroller. *Device Datasheet*, Revised 2011.

[46] Xabier Iturbe, Balaji Venu, Emre Ozer, and Shidhartha Das. A Triple Core Lock-Step (TCLS) ARM Cortex-R5 Processor for Safety-Critical and Ultra-Reliable Applications. *IEEE/IFIP International Conference on Dependable Systems and Networks Workshops*, 2016.

[47] Gregory H. Johnson, Jakob H. Hohl, Ronald D. Schrimpf, and Kenneth F. Galloway. Simulating Single-Event Burnout of n-Channel Power MOSFET's. *IEEE Transactions On Electron Devices*, 1993.

[48] Jet Propulsion Laboratory. TOPEX/Poseidon Fact Sheet. `https://sealevel.jpl.nasa.gov/missions/topex/topexfactsheet/,`. [Online; accessed September 2020].

[49] Jet Propulsion Laboratory. Missions: TOPEX/Poseidon. `https://sealevel.jpl.nasa.gov/missions/topex/,`. [Online; accessed September 2020].

[50] Kenneth A. LaBel and Michele M. Gates. Single-Event-Effect Mitigation from a System Perspective. *IEEE Transactions On Nuclear Science*, 1996.

[51] Jet Propulsion Laboratory. Mission to comet Tempel 1: Deep Impact. `https://www.jpl.nasa.gov/missions/deep-impact/,`. [Online; accessed September 2020].

[52] The Johns Hopkins University Applied Physics Laboratory. New Horizons: Systems and Components. `http://pluto.jhuapl.edu/Mission/Spacecraft.php#Systems-and-Components,`. [Online; accessed September 2020].

[53] The Johns Hopkins University Applied Physics Laboratory. Spacecraft Systems and Components. `http://pluto.jhuapl.edu/Mission/Spacecraft.php#Systems-and-Components,`. [Online; accessed September 2020].

[54] Mukesh Lahori and Kausar Johar. SW-based, Run-time Diagnostic Tests to Minimize System-level Effort. *ARM TechCon*, 2018.

[55] Tuvia Liran, Ran Ginosar, Dov Alon, Reuven Dobkin, and Michael Goldberg. Next generation RadSafe technology for SoCs. *Data Systems in Aerospace*, 2012.

[56] Joseph R. Marshall and Richard W. Berger. Processor Solution For The Second Century Of Powered Space Flight. *19th Digital Avionics Systems Conference Proceedings*, 2000.

[57] ARM MBED. WatchDog Timer. `https://os.mbed.com/cookbook/WatchDog-Timer`. [Online; accessed September 2020].

[58] Kevin McManamon, Richard Lancaster, and Nuno Silva. EXOMARS Rover Vehicle Perception System Architecture And Test Results. *ASTRA Conference*, 2013.

[59] Mentor. Questa Advanced Simulator. `https://www.mentor.com/products/fv/questa/`. [Online; accessed September 2020].

[60] T.F. Miyahira, A.H. Johnston, H.N. Becker, S.D. LaLumondiere, and S.C. Moss. Catastrophic Latchup in CMOS Analog-to-Digital Converters. *IEEE Transactions On Nuclear Science*, 2001.

[61] Keith S. Morgan, Daniel L. McMurtrey, Brian H. Pratt, and Michael J. Wirthlin. A Comparison of TMR With Alternative Fault-Tolerant Design Techniques for FPGAs. *IEEE Transactions On Nuclear Science*, 2007.

[62] I. Mouret, P. Calvel, M. Allenspach, J.L. Titus, C.F. Wheatley, K.A. LaBel, M.C. Calvet, R.D. Schrimpf, and K.F. Galloway. Measurement of a Cross-Section for Single-Event Gate Rupture in Power MOSFET's. *IEEE Electron Device Letters*, 1996.

[63] Mahsa Mousavi, Sayandip De, Hamid Reza Pourshaghaghi, and Henk Corporaal. Fault Tolerant FPGAs: Where to Spend the Effort? *22nd Euromicro Conference on Digital System Design (DSD)*, 2019.

[64] N2YO.com. ENVISAT. `https://www.n2yo.com/satellite/?s=27386#results`. [Online; accessed September 2020].

[65] NASA. Kepler and K2. `https://www.nasa.gov/mission_pages/kepler/main/index.html`, . [Online; accessed September 2020].

[66] NASA. Single Event Effects Specification. `https://radhome.gsfc.nasa.gov/radhome/papers/seespec.htm`, . [Online; accessed September 2020].

[67] NASA. Mars 2020 Mission: Mission Facts. `https://mars.jpl.nasa.gov/mars2020/`, . [Online; accessed September 2020].

[68] NASA. Mars 2020 Mission: Brains. `https://mars.jpl.nasa.gov/mars2020/mission/rover/brains/`, . [Online; accessed September 2020].

[69] NASA. Radiation Belt Storm Probes Depicted in their Orbit . `https://www.nasa.gov/mission_pages/sunearth/news/gallery/rbsp-donuts.html`, . [Online; accessed September 2020].

[70] NASA. New Horizons. `https://www.nasa.gov/mission_pages/newhorizons/main/index.html`, . [Online; accessed September 2020].

[71] NASA/JPL-Caltech/SwRI. Sources of Ionizing Radiation in Interplanetary Space. `https://www.nasa.gov/mission_pages/msl/multimedia/pia16938.html`. [Online; accessed September 2020].

[72] Eugene Normand. Single Event Upset at Ground Level. *IEEE Transactions On Nuclear Science*, 1996.

[73] GPS official webpage. Space Segment. `https://www.gps.gov/systems/gps/space/`. [Online; accessed September 2020].

[74] Timothy R. Oldham. Basic Mechanisms of TID and DDD Response in MOS and Bipolar Microelectronics. *IEEE Nuclear and Space Radiation Effects Conference*, 2011.

[75] T.R. Oldham and F.B. McLean. Total Ionizing Dose Effects in MOS Oxides and Devices. *IEEE Transactions On Nuclear Science*, 2003.

[76] Janak H. Patel. Stuck-At Fault: A Fault Model for the next Millennium?, howpublished = "`https://web.stanford.edu/class/ee386/public/stuck_at_fault_6per_page`", note = "[online; accessed september 2020]".

[77] NASA Mars Exploration Program. Mars Curiosity Rover. `https://mars.jpl.nasa.gov/msl/mission/overview/`, . [Online; accessed September 2020].

[78] NASA Mars Exploration Program. Mars Curiosity Rover: Brains. `https://mars.jpl.nasa.gov/msl/spacecraft/rover/brains/`, . [Online; accessed September 2020].

[79] Jan M. Rabaey, Anantha Chandrakasan, and Borivoje Nikolic. Digital Integrated Circuits: A Design Perspective. *Pearson Education*, 2003.

[80] R.D. Schrimpf. Radiation Effects in Microelectronics. *Radiation Effects on Embedded Systems*, 2007.

[81] James R. Schwank, Marty R. Shaneyfelt, Daniel M. Fleetwood, James A. Felix, Paul E. Dodd, Philippe Paillet, and Véronique Ferlet-Cavrois. Radiation Effects in MOS Oxides. *IEEE Transactions On Nuclear Science*, 2008.

[82] NASA Science. Mars Curiosity Rover - Brains. `https://mars.nasa.gov/msl/spacecraft/rover/brains/`. [Online; accessed September 2020].

[83] David Selčan, Gregor Kirbiš, and Iztok Kramberger. Nanosatellites in LEO and beyond: Advanced Radiation protection techniques for COTS-based spacecraft. *Acta Astronautica*, 2016.

[84] SPENVIS. Space Environment Information System. `https://www.spenvis.oma.be/`. [Online; accessed September 2020].

[85] Synopsys. Z01X Functional Safety Assurance. `https://www.synopsys.com/verification/simulation/z01x-functional-safety.html`. [Online; accessed September 2020].

[86] Synopsys. Z01X Simulator. *Safety Verification User Guide*, 2020.

[87] SYNOVA. Mongoose-V MIPS R3000 Rad-Hard Processor: Product Summary. `https://web.archive.org/web/20151121100152/http://synova.com/proc/mg5.html`, . [Archived Online; accessed September 2020].

[88] SYNOVA. Mongoose-V Prices (per unit). `https://web.archive.org/web/20151031064024/http://synova.com/proc/mg5_price.html`, . [Archived Online; accessed September 2020].

[89] SYNOVA. Mongoose-V 32-bit MIPS Microprocessor: Architecture Description. *Device Datasheet*, Revised 1997.

[90] BAE Systems. RAD750 radiation-hardened PowerPC microprocessor. *Device Datasheet*, Revised 2008.

[91] BAE Systems. Radiation-hardened electronics product guide. *Product Catalog*, Revised 2017.

[92] Microchip Technology. AT91SAM9G20. `https://www.microchip.com/wwwproducts/en/AT91SAM9G20`, . [Online; accessed September 2020].

[93] Microchip Technology. PIC24FJ256GA110, General Purpose 256 KB Flash Microcontroller. `https://www.microchip.com/wwwproducts/en/PIC24FJ256GA110`, . [Online; accessed September 2020].

[94] Microchip Technology. 32-bit ARM-Based Microprocessors: SAM9G20. *Device Datasheet*, Revised 2015.

[95] Microchip Technology. PIC24FJ256GA110 Family: 64/80/100-Pin, 16-Bit, General Purpose Flash Microcontrollers with Peripheral Pin Select (PPS). *Device Datasheet*, Revised 2019.

[96] Wikipedia. Hamming code. `https://en.wikipedia.org/wiki/Hamming_code`, . [Online; accessed September 2020].

[97] Wikipedia. Apollo Guidance Computer. `https://en.wikipedia.org/wiki/Apollo_Guidance_Computer`, . [Online; accessed September 2020].

[98] Wikipedia. Envisat. `https://en.wikipedia.org/wiki/Envisat`, . [Online; accessed September 2020].

[99] Wikipedia. Curiosity (rover). `https://en.wikipedia.org/wiki/Curiosity_(rover)`, . [Online; accessed September 2020].

[100] Wikipedia. New Horizons. `https://en.wikipedia.org/wiki/New_Horizons#Flight_computer`, . [Online; accessed September 2020].

[101] Wikipedia. RAD750. `https://en.wikipedia.org/wiki/RAD750`, . [Online; accessed September 2020].

[102] Xilinx. Zynq-7000 SoC Data Sheet: Overview. *Device Datasheet*, Revised 2018.