UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM MICROELETRÔNICA

ÁDRIA BARROS DE OLIVEIRA

Applying Dual-Core Lockstep in Embedded Processors to Mitigate Radiation-induced Soft Errors

Thesis presented in partial fulfillment
of the requirements for the degree of
Master of Microeletronics

Advisor: Prof[a]. Dr[a]. Fernanda Lima Kastensmidt

Porto Alegre
November 2017

CIP — CATALOGAÇÃO NA PUBLICAÇÃO

Oliveira, Ádria Barros de

Applying Dual-Core Lockstep in Embedded Processors to Mitigate Radiation-induced Soft Errors / Ádria Barros de Oliveira. – Porto Alegre: PGMICRO da UFRGS, 2017.

95 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Microeletrônica, Porto Alegre, BR–RS, 2017. Advisor: Fernanda Lima Kastensmidt.

1. Embedded Processors Reliability. 2. Fault Tolerance. 3. Lockstep. 4. Soft Errors. 5. Radiation Experiments. 6. Fault Injection. I. Kastensmidt, Fernanda Lima. II. Applying Dual-Core Lockstep in Embedded Processors to Mitigate Radiation-induced Soft Errors.

“Let us pick up our books and our pens. They are our most powerful weapons. One child, one teacher, one book, and one pen can change the world. Education is the only solution.”

— MALALA YOUSAFZAI

ACKNOWLEDGMENT

I would like to thanks my parents, Dinancy and Adons, that have always taken care of me. I am grateful for all support and learning that allowed me to be who I am. Thank you for believing in me even when I thought that I was not capable. Thank you for all unconditional love.

Jeckson, thank you for all love and support, mainly on the hard days. Thank you for being my safe harbor, for being at my side and do not give up on me. You are so much more than a boyfriend. You are my love, my partner, and my friend, and I am so thankful for that. Thank you for making me so happy. I love you.

Gennaro, Ygor, and Filipe I would like to thanks for all you have done for me. I really appreciate our friendship.

Many thanks to all my friends that even far supported me. “Friends are the family that we choose for ourselves”, and I chose you all.

I would like to express my special thanks of gratitude to my advisor Fernanda who gave me the opportunity to work on this project. Thank you for trusting in me and for all your dedication and guidance. You are an inspiration for me.

A heartfelt thank you to all.

ABSTRACT

The embedded processors operating in safety- or mission-critical systems are not allowed to fail. Any failure in such applications could lead to unacceptable consequences as life risk or significant damage to property or environment. Concerning faults originated by the radiation-induced soft errors, the embedded systems operating in aerospace applications are particularly susceptible. However, the radiation effects can also be observed at ground level. Soft errors affect processors by modifying values stored in memory elements, such as registers and data memory. These faults may lead the processor to execute an application incorrectly, generating output errors or leading hangs and crashes in the system. The recent advances in embedded systems concern the integration of hard-core processors and FPGAs. Such devices, called All Programmable System-on-Chip (APSoC), are also susceptible to radiation effects. Aiming to address this fault tolerance problem this work presents a Dual-Core LockStep (DCLS) as a fault tolerance technique to mitigate radiation-induced faults affecting processors embedded into APSoCs. Lockstep is a method based on redundancy used to detect and correct soft errors. The proposed DCLS is implemented in a hard-core ARM Cortex-A9 embedded into a Zynq-7000 APSoC. The approach efficiency was validated not only on applications running in bare-metal but also on top of FreeRTOS systems. Heavy ions experiments and fault injection emulation were performed to analyze the system susceptibility to bit-flips. The obtained results show that the approach is able to decrease the system cross section with a high rate of protection. The DCLS system successfully mitigated up to 78% of the injected faults. Software optimizations were also evaluated to understand the trade-offs between performance and reliability better. By the analysis of different software partitions, it was observed that the execution time of an application block must to be much longer than the verification time to achieve fewer performance penalties. The compiler optimizations assessment demonstrate that using O3 level increases the application vulnerability to soft errors. Because O3 handles more registers than other optimizations, the system is more susceptible to faults. On the other hand, results from radiation experiments show that O3 level provides a higher Mean Workload Between Failures (MWBF). As the application runs faster, more data are correctly computed before an error occurrence.

Keywords: Embedded Processors Reliability. Fault Tolerance. Lockstep. Soft Errors. Radiation Experiments. Fault Injection.

Aplicando Dual-Core Lockstep em Processadores Embarcados para Mitigar Falhas Transientes Induzidas por Radiação

RESUMO

Os processadores embarcados operando em sistemas de segurança ou de missão crítica não podem falhar. Qualquer falha neste tipo de aplicação pode levar a consequências inaceitáveis, como risco de vida ou danos à propriedade ou ao meio ambiente. Os sistemas embarcados que operam em aplicações aeroespaciais são sucetíveis à falhas transientes induzidas por radiação. Entretanto, os efeitos de radiação também podem ser observados ao nível do solo. Falhas transientes afetam os processadores modificando os valores armazenados em elementos de memória, tais como registradores e memória de dados. Essas falhas podem levar o processador a executar incorretamente a aplicação, provocando erros na saída ou travamentos no sistema. Os avanços recentes em processadores embarcados concistem na integração de processadores hard-core e FPGAs. Tais dispositivos, comumente chamados de Sistemas-em-Chip Totalmente Programáveis (APSoCs), também são sucetíveis aos efeitos de radiação. Com objetivo de minimizar esse problema de tolerância a falhas, este trabalho apresenta um Dual-Core LockStep (DCLS) como uma técnica de tolerância para mitigar falhas induzidas por radiação que afetam processadores embarcados em APSoCs. Lockstep é um método baseado em redundância usado para detectar e corrigir falhas transientes. O DCLS proposto é implementado em um processador ARM Cortex-A9 hard-core embarcado no APSoC Zynq-7000. A eficiência da abordagem implementada foi validada tanto em aplicações executando em bare-metal como no sistema operacional FreeRTOS. Experimentos com íons pesados e emulação de falhas por injeção foram executados para analisar a sucetibilidade do sistema a inversão de bits. Os resultados obtidos mostram que a abordagem é capaz de diminuir a seção de choque do sistema com uma alta taxa de proteção. O sistema DCLS mitigou com sucesso até 78% das falhas injetadas. Otimizações de software também foram avaliadas para uma melhor compreenção dos trade-offs entre desempenho e confiabilidade. Através da análise de diferentes partições de software, observou-se que o tempo de execução de um bloco da aplicação deve ser muito maior que o tempo de verificação para que se obtenha menor impacto em desempenho. A avaliação de otimizações de compilador demonstrou que utilizar o nível O3 aumenta a vulnerabilidade da aplicação à falhas transientes. Como o O3 requer o uso de mais registradores que os otros níveis de otimização, o sistema se torna mais sucetí-

vel à falhas. Por outro lado, os resultados dos experimentos de radiação apontam que a aplicação compilada com nível O3 obtém maior Carga de Trabalho Média Entre Falhas (MWBF). Como a aplicação executa mais rápido, mais dados são computados corretamente antes da ocorrência de um erro.

Palavras-chave: Confiabilidade de processadores embarcados, Tolerância a falhas, Lockstep, Falhas Transientes, Experimentos de radiação, Injeção de Falhas.

LIST OF ABBREVIATIONS AND ACRONYMS

AES	Advanced Encryption Standard
APSoC	All Programmable System-on-Chip
BRAM	Block RAM
CPU	Central Processing Unit
CRT	Chip-level Redundant Threading
CRTR	Chip-level Redundantly Threaded multiprocessor with Recovery
COTS	Commercial Off The Shelf
CFC	Control Flow Checking
DCLS_BR_DDR	DCLS accessing BRAM and DDR memories
DCLS_BR	DCLS accessing BRAM memory only
DUT	Device Under Test
DSP	Digital Signal Processor
DCLS	Dual-Core LockStep
DMR	Dual Modular Redundancy
DWC	Duplication With Comparison
DCC	Dynamic Core Coupling
ESA	European Space Agency
ECC	Error Correction Code
EDAC	Error Detection and Correction
EDC	Error Detection Code
FPGA	Field Programmable Gate Array
FSM	Finite-State Machine
FF	Flip-Flop
FPU	Floating Point Unit

GPU	Graphics Processing Unit
HDL	Hardware Description Language
HPC	High Performance Computing
HETA	Hybrid Error-detection Technique using Assertions
IU	Integer Unit
IP	Intellectual Property
LAFN-USP	Laboratório Aberto de Física Nuclear at Universidade de São Paulo
LET	Linear Energy Transfer
LR	Link Register
LANL	Los Alamos National Laboratory
LUT	Look-Up Table
MxM	Matrix Multiplication
MWBF	Mean Workload Between Failures
MBU	Multiple Bit Upset
MPSoC	Multiprocessor System-on-Chips
NMR	N-modular redundancy
OCM	On-Chip Memory
OS	Operating Systems
PS	Processing System
PC	Program Counter
PL	Programmable Logic
Rad-Hard	Radiation Hardened
RTL	Resistor-Transistor Logic
S-SETA	Selective Software-only Error-detection Technique using Assertions
SDC	Silent Data Corruption
SRT	Simultaneous and Redundantly Threaded

SMT	Simultaneous Multithreading
SRTR	Simultaneously and Redundantly Threaded processors with Recovery
SBU	Single Bit Upset
SEE	Single Event Effect
SEFI	Single Event Functional Interrupt
SEL	Single Event Latch-up
SEU	Single Event Upset
SET	Single Event Transient
SRMT	Software-based Redundant Multi-Threading
SIHFT	Software-Implemented Hardware Fault Tolerance
SP	Stack Pointer
SoC	System-on-Chip
TLR	Thread-Level Redundancy
TMR	Triple Modular Redundancy
VP	Verification Point

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

State-of-the-art computing systems rely on heterogeneous architectures to achieve performance and energy consumption goals. Dependable and safety-critical systems are no exception to this rule. In the last years, new embedded Systems-on-Chips (SoC) based on heterogeneous architectures have been developed to address those requirements. These SoCs, called All Programmable System-on-Chip (APSoC), combine a Field Programmable Gate Array (FPGA) layer with embedded processors. Frequently, commercial APSoCs use SRAM-based FPGA solutions due to the high reconfiguration flexibility, competitiveness costs, and capability of integrating complex systems on the same component. Unfortunately, whilst being able to achieve high performance and energy consumption with low cost, the APSoCs devices are subject to a plethora of issues that may compromise their usage for safety-critical and dependable purposes.

The Safety- and mission-critical applications are not allowed to fail. Aerospace, nuclear, medical and automotive are examples of such systems, where any failure could lead to unacceptable consequences as life risk or significant damage to property or environment. Therefore, protection against faults that provoke errors is demanded. As mentioned, the APSoCs can be suitable for such applications, but the system must be protected with techniques to ensure high reliability.

Concerning faults originated by the radiation-induced soft errors, the embedded systems operating in aerospace applications are particularly susceptible. However, the radiation effects can also be observed at ground level due to interaction with neutron particles present in the atmosphere. Massive doses of radiation can cause several execution problems (QUINN, 2014). With the reduction of the transistor size, modern processors became more susceptible to Single Event Effects (SEE) (BAUMANN, 2005). The particles can interact with silicon, provoking transient pulses in some sensitive areas. Such episodes might lead to Single Event Upset (SEU) – or bit flips – in the sequential logic that could induce errors, generating wrong application's results and other failures in the system, like hangs and crashes (AZAMBUJA; KASTENSMIDT; BECKER, 2014). When the radiation is high enough, it may flip data bits of memory cells, registers, latches, and flip-flops that can lead to errors (DODD et al., 2003). Radiation effects also induce faults in FPGAs. The ones that use SRAM-based technologies are very susceptible to soft errors. As they are composed of millions of SRAM cells used to configure all the synthesized logic, the embedded processors, Digital Signal Processors (DSP), and memories

(SIEGLE et al., 2015). Because APSoCs are not architecturally fault tolerant, as they can be composed of multiple processors cores, Graphics Processing Units (GPU) and a programmable array that are all susceptible to radiation, safety-critical systems making use of that hardware shall present fault tolerance mechanisms.

Several methods have been described in the literature to deal with those radiation-caused errors (MAHMOOD; MCCLUSKEY, 1988; REINHARDT; MUKHERJEE, 2000; GOLOUBEVA et al., 2006; MUKHERJEE; KONTZ; REINHARDT, 2002). Most of the fault tolerance methods protect the system from Silent Data Corruptions (SDC), which are errors in memory and final application output. Single Event Functional Interrupt (SEFI), another important type of error that leads to hangs or crashes in the system, is typically not targeted by the techniques. Nevertheless, SEFI is the most severe issue, as it impacts the application execution and the user experience directly.

There are just a few techniques that are able to detect both SDC and SEFI. One of them employs the lockstep principle, which is a hybrid solution based on redundancy used to increase the dependability of embedded processors. This work makes use of Dual-core Lockstep (DCLS) as a fault tolerance method capable of detecting and correcting soft errors. The proposed DCLS runs the same application simultaneously in two identical processors and monitors the application execution outputs, checking for inconsistencies. The lockstep combines checkpoints and rollbacks (BOWEN; PRADHAM, 1993) at software level to ensure processor reliability. Checkpoints are operations that store a fault-free copy of the processor state in a safe memory. Whereas, a rollback consists of a recovery method that restores the processor to a previous safe state.

Although the literature presents a plethora of lockstep implementations (ABATE; STERPONE; VIOLANTE, 2008; VIOLANTE et al., 2011; GOMEZ-CORNEJO et al., 2013; PHAM; PILLEMENT; PIESTRAK, 2013), most of them protect soft-core processors. As these cores have an open architecture and are implemented in FPGA logic, the advantage is the designer can modify it if needed. However, the ones embedded into SRAM-based FPGAs are susceptible to persistent soft errors in the configuration bitstream, which can affect the functionality, and performance of the processor. This work focuses on protecting hard-core processors embedded into APSoCs. Therefore, the persistent errors in the processors caused by bit-flips in the programmable configurable blocks are avoided, differently as occurs with soft-core processors. Moreover, it is possible to take advantages of the high-performance hard-core processor as well as the programmable matrix.

1.1 Main Objective and Contributions

This work aims to investigate a fault tolerant solution to soft-errors mitigation concerning processors embedded into APSoCs. As previously discussed, heterogeneous architectures can efficiently execute applications, achieving performance enhancement. However, these devices are very susceptible to radiation effects been necessary to implement strategies to deal with it.

The main objective of this work is implementing an approach based on Dual-Core LockStep (DCLS) as a fault tolerance technique to mitigate radiation-induced faults affecting processors embedded into APSoCs. The method should be capable of detecting and treating both SDC and SEFI errors before they can cause catastrophic issues.

The proposed DCLS architecture relies on two processors running with private embedded BRAM memories to duplicate the application execution, and a checker module to validate the processors' output and, in case of failure, recover the system. The DCLS technique is implemented to protect a dual-core ARM Cortex-A9 embedded into a Xilinx Zynq-7000 APSoC. This APSoC is capable of serving a wide range of safety-critical applications, such as avionics communications, missile control, and smart ammunition control systems. The Zynq was chosen as test case due to its high presence on the market and in the scientific literature. Although the implementation target is the ARM-A9 into Zynq, the approach can be easily portable to another hard-core processor embedded into APSoC, making just some architectural changes.

The main novelty lies in applying the DCLS to a hard-core ARM Cortex-A9 in which, for the best of our knowledge, no other work focuses on this processor. Besides, the use of an exclusive BRAM memory to each processor, in order to avoid a single point of failures in the data memory, increasing the reliability. The implemented lockstep approach does not require both processors work in the same clock, because of the verification point strategy each processor can work with distinct clock. This work applies the DCLS to applications running not only in bare-metal but also on the FreeRTOS, which is an industry-leading real-time operating system. The FreeRTOS is used in a plethora of applications, such as aerospace and safety-critical. Thus, the effectiveness evaluation of fault-tolerant techniques applied to applications running on top of this systems is essential.

A complete evaluation of the performance impact in used the proposed DCLS in the system is described. This work presents an analysis of how software optimizations

can affect the DCLS approach fault coverage and performance penalties. Two types of optimization are evaluated: software partition and compiler optimizations. The former is directly related to the number of verification points, which is defined during the DCLS design. The latter is related to compilation parameters used to boost performance.

To assess the system, two types of experiments were performed: radiation exposure and fault injection. The radiation exposure consists of heavy ions experiments, conducted at Laboratório Aberto de Física Nuclear at Universidade de São Paulo (LAFN-USP), Brazil. To emulate faults in the processor a fault injection system was developed. The injection is performed by a module in the FPGA layer of the APSoC that emulates bit-flips on the processor's register file in a non-intrusive manner. Thus, the resilience of the embedded processor is evaluated.

The appendix A presents the list of publications resulting from this work efforts.

1.2 Work Structure

This dissertation is organized as follows. Chapter 2 presents the background knowledge. The basics concepts of embedded processors, heterogeneous architectures and radiation effects are introduced. Besides, an overview of fault tolerant techniques used to improve the processor dependability is presented. Then, in chapter 3 is described the implementation of the proposed DCLS technique. In sequence, chapter 4 details the evaluation methodology used to assess the system. It describes the experiments performed and test case applications used for achieving the results, which are discussed in chapter 5. Finally, chapter 6 draws the final remarks and possible future works that can be done over the presented DCLS.

2 BACKGROUND AND STATE OF THE ART

This chapter first discusses embedded processors and how soft errors can influence in their functionality. The system complexity and the area density increase the vulnerability to soft errors. Devices with heterogeneous architectures, which integrate embedded processors and FPGAs, are also described. These devices are extensively being used for implementing mission-critical and reliable systems. Such applications demand fault tolerant techniques capable of protecting the system against soft errors. An overview of fault tolerant techniques used to improve the processor dependability is presented, and their advantages and disadvantages are discussed. Finally, the concepts of the lockstep method and several related works are presented in the last section.

2.1 Embedded Processors

Considering the evolution of the architectures, the modern processors delivers high-performance computing, power efficiency, and reduced cost, which are requirements for almost all safety-critical applications. From the first Von Neumann machines up to high-performance processors, the technology advances allowed to rise from sequential execution of instructions to a high level of parallelism.

The architectures of embedded processors can be split into three types depending on the number of Central Processing Units (CPU): single-, multi-, or many-cores. In the former, there is only one CPU, while the multi-cores are usually composed by two, four, six, or eight independent cores in the same silicon. The many-core system is defined when ten or more CPUs are integrated. In single-core processors to run multiple programs, time slices are assigned. In multi- and many-cores, multiple tasks that can be run in parallel at the same time, boosting performance (JOHNSON; DINYO, 2015).

The embedded processors can be defined as hard- or soft-cores. The hard-cores are processors designed physically in the chip, where all the components are integrated and manufactured. Thus, the processor is hardwired in the die, like Power-PC and ARM Cortex-A9. By contrast, the soft-cores are the ones implemented using reconfigurable resources of FPGAs. These processors are Intellectual Property (IP) blocks developed entirely in Hardware Description Language (HDL), like VHDL or Verilog. The MicroBlaze, PicoBlaze, and Nios are examples of soft-core processors used in many applications. Moreover, there are those proposed by the open-source community, like LEON3

and LEON4. The soft-cores main advantages concern to flexibility, simple integration, device customization, reusability, and portability. However, the soft-cores usually are slower than hard-cores. Besides, the soft-cores are not energy efficient, while hard-cores possess power saving modes (MONDRAGON, 2012).

2.2 Software Optimizations

Aiming to achieve performance requirements, less memory occupation and decrease the power consumption, the program executing in a processor can be optimized in some ways. Different levels can be optimization target, like assembly, compile, build, source code, algorithms and data structures, and design level (LINS, 2017). Software optimizations can be characterized as architecture-independent and architecture-dependent (DOMEIKA, 2008). The latter uses specific properties of the architecture, changing parameters depending on the underlying platform, like register allocation and instruction scheduling. On the other hand, architecture-independent optimizations are more general techniques that can be applied to different platforms, like loop-invariant code motion, dead code elimination, and common-subexpression elimination.

The optimizing compiler applies some transformations to the program aiming to get a more efficient code. The optimization must not affect the application results. However, it is allowed to change the program flow. Fundamentally, the compiler can optimize the code by control and data flow analysis (HAGEN, 2006). To assess the control flow, the compiler examines control constructs and loops to determine the paths of execution that can be taken and simplify the code. The data flow analysis concerns the program verification for the usage track of the variables, allowing to apply the optimizations.

Distinct compiler optimization levels can be applied to the code, with slightly different set of options. Depending on the selected configuration, the focus can be performance and/or code size improvement. The GCC optimization levels are described below (STALLMAN; COMMUNITY, 2014).

- -O0: This is the primary level, in which no optimization is performed. The source code is compiled without any rearrangement, and the executable instructions correspond precisely to the program. The -O0 is default level in debug mode.
- -O1: This is the optimization level one, which has restricted optimizations. This level focuses in to reduce code size and execution time, without compromise the

compilation time.

- -O2: This is the default level in release mode. Almost all the optimizations are performed by the compiler, less the space-speed tradeoff ones. The compiler attempt to improve the performance at the expense of compilation time.
- -O3: This is the highest level, in which all the optimizations are enabled. The compiler focuses on boosting performance over code size cost.

2.3 Embedded Operating Systems

Several embedded computing systems use Operating Systems (OS) due to efficiency and software scalability. Differently from the OS for a desktop computer, the embedded OS typically requires resource-efficiency and reliability. Also, the embedded OSes have to be able to run with memory and processing power constraints. Due to the wide variety of embedded applications, there are a large number of available OS options, like Linux, FreeRTOS, and eCos.

For real-time systems, the FreeRTOS (BARRY, 2017) is the industry-leading operating system. FreeRTOS is a light OS that has, as main advantage, the feasibility of working with threads and pseudoparallelism on a single-core processor. Accordingly, the main application can be split into individual tasks with assigned priorities to be performed by the processor.

The main advantages in using embedded OS in the systems are the possibility to customize the OS to precise application requirements, the efficiency improvement, and the scalability. Despite these advantages, as the system complexity is increased, more critical points of failure and susceptibility to errors are observed, which becomes a concern for safety-critical applications (RODRIGUES; KASTENSMIDT, 2017). Thus, the use of embedded OS in these applications requires the implementation of fault-tolerance techniques.

2.4 Radiation Effects

The progress of the semiconductor industry made possible the development of more complex systems, allowing architectures integration and increasing systems functionalities. The new technologies provide transistors with reduced dimensions, enabling

to add a high number of transistors per area unit. Besides, the advances lead to tightened noise margins, and threshold voltages and node capacitances reduced. All these advances made the devices more susceptible to the radiation effects (BAUMANN, 2005).

Embedded systems operating in aerospace applications are particularly susceptible to radiation-induced soft errors caused by ionized particles originated from the spatial environment. The particles can interact with silicon, depositing enough charge that can temporarily charge or discharge the transistors drain at off-state, provoking transient pulses in the susceptible nodes. Such episodes might lead to bit-flips in the sequential logic that later on can induce to errors and failures in the system (BAUMANN, 2005). Systems in avionics and at ground level can also be affected by radiation effects due to interaction with neutrons present in the atmosphere (NORMAND, 2001). Baumann (2005) defines three primary mechanisms responsible for soft errors at terrestrial altitudes. The first is the interaction between high-energy neutrons and silicon nucleus. The second is when high concentrations of the dopant boron present in the device react with low-energy neutrons. And the last one is the emission of alpha particles, composed of two neutrons and two protons, from radioactive impurities in the materials.

The pulses of transient voltage caused by radiation disturbances are called as Single Event Effects (SEE), which can be destructive or nondestructive (BAUMANN, 2005; SIEGLE et al., 2015). Single Event Latch-up (SEL) is a destructive effect, as result of a high operating current. The nondestructive ones can be classified as Single Event Upset (SEU), Single Event Functional Interrupt (SEFI) or Single Event Transient (SET). SET is the transient pulse generated at a susceptible transistor node of design. This pulse may propagate through the logic and be captured by a flip-flop, for instance. SET can be logical, electrical, or latch window masked. A SEFI occurs when a soft error leads to malfunction of the device. The SEU is described as a radiation effect that provokes charge disturbance enough to reverse a data state of a memory element. When only one memory bit is affected by the soft error, it is called Single Bit Upset (SBU). If two or more bits are flipped in the same memory word, this event is defined as Multiple Bit Upset (MBU).

2.4.1 Faults, Errors and Failures Concepts

The definition of faults, errors and failures can present small differences in the literature. This work adopts the same concepts described in (AVIZIENIS et al., 2004). The difference between the implemented hardware and its functionality is called a defect

or upset. Whereas, the logical abstraction of a physical defect, or imperfection, is defined as a fault. However, if the defect is electrically masked the fault will not be observed. The faults can be classified as permanent, transient or intermittent. The first are the ones that are continuous in time and remain until the replacement (or reparation) of the component. The second and third are temporary faults that are time limited. The transient ones occur randomly and remain during a short period. A repetitive fault is classified as intermittent.

When a fault is manifested, it can lead to an error, which is a wrong value in the output. In other words, an error is an inconsistency in the final result. If the error propagates and affects the system externally, it is said that a failure occurs. Therefore, an upset can be manifested as a fault that could lead to an error that may be propagated generating a failure. It is important to notice that not all faults result in error, and not every error produces a failure.

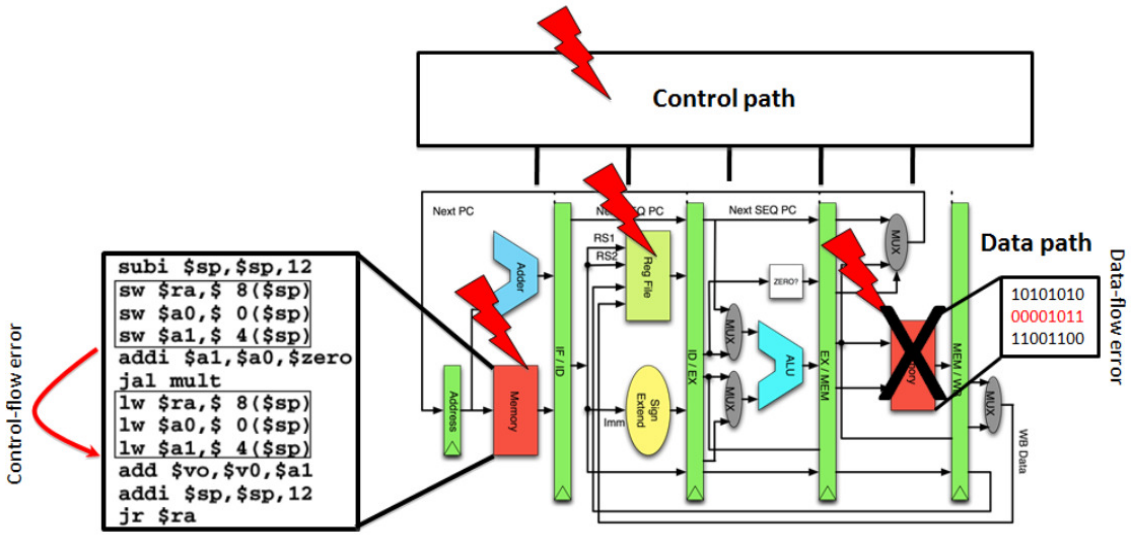
2.4.2 Single Event Upsets in Embedded Processors

Soft errors in processors may lead to executing an application incorrectly, generating data errors or system crashes. As Fig. 2.1 shows, SEUs can affect the data flow or control flow of the processor (TAMBARA, 2017).

Upsets in the values stored in memory elements can lead to data flow errors, which can be caused by incorrect operations or wrong data. The execution of an incorrect operation occurs when a bit-flip corrupts the program code, leading to a wrong instruction. If the bit-flip affects a data used as input by an operation, the output from it certainly will be incorrect. In both data flow errors, the program output is corrupted, leading to wrong results in the final application that are classified as Silent Data Corruption (SDC).

When an SEU affects the control flow, the processor may execute the program incorrectly, leading to SEFIs. In this case, the SEFI is caused by an application crash and processor hang. Upsets in the control flow may lead to branch errors: creation or deletion of a branch, or incorrect branch decision. The erroneous creation is due to a bit-flip that set a non-branch instruction into a branch, leading the program flow to a wrong address. The erroneous deletion occurs when a branch is not taken when it should be because the branch instruction was converted into another instruction. A bit-flip in a conditional branch can result in an incorrect decision, i.e. if the branch should be taken or not. Also, an SEU can modify the register that contains the target address of a branch instruction, assigning an incorrect address to the program execution. Finally, when the Program Counter (PC)

Figure 2.1: Possible effects of soft errors in processors



Source: (TAMBARA, 2017).

register is affected by a soft error, the next instruction to be executed changes, which also leads the program flow to a wrong address.

As presented, the SEUs may lead to different types of errors in the processors, depending on the memory element affected. Velazco and Faure (2007) relate the resulted error with the affected hardware unit, as described below:

- **Register File:** Upsets may corrupt data, provoking errors in the application outputs (results inconsistency). If an SEU affects a control register, this can lead to errors in the program execution flow and hangs in the system.
- **Integer Unit (IU), Floating Point Unit (FPU):** Due to the pipeline in the arithmetic units, SEUs may result in incorrect computations.
- **Bus Unit:** Bit-flips in the embedded registers, which latches address and data, can lead to incorrect read or write operations.
- **Control Unit:** SEUs in the circuitry, which implements complex algorithms, may provoke exception generation or losses of sequence.
- **Debug Unit:** The special execution modes can be trigged by SEUs, which will lead to errors in the execution flow of the program.
- **Instruction Cache:** Upsets may lead to corrupted outputs or processor hangs. The instruction caches are usually divided into SRAM array to store fetched instructions and tag array to validate or invalidate the fetched program. SEUs in the tag array can invalidate an instruction to be executed, leading to a cache miss, which introduces a

delay in the program execution as the instruction has to be fetched again. However, if an SEU validates an incorrect code, the flow of the program will be crashed. Besides, an upset can affect the SRAM array and corrupts an instruction. If the tag array validates the corrupted code, a wrong instruction will be executed, or an exception will be generated. The latter occurs when the corrupted instruction is not anymore in the processor instruction set. On the other hand, if the corrupted instruction is not validated by the tag array, the fault is masked, and no incorrect behavior will be observed.

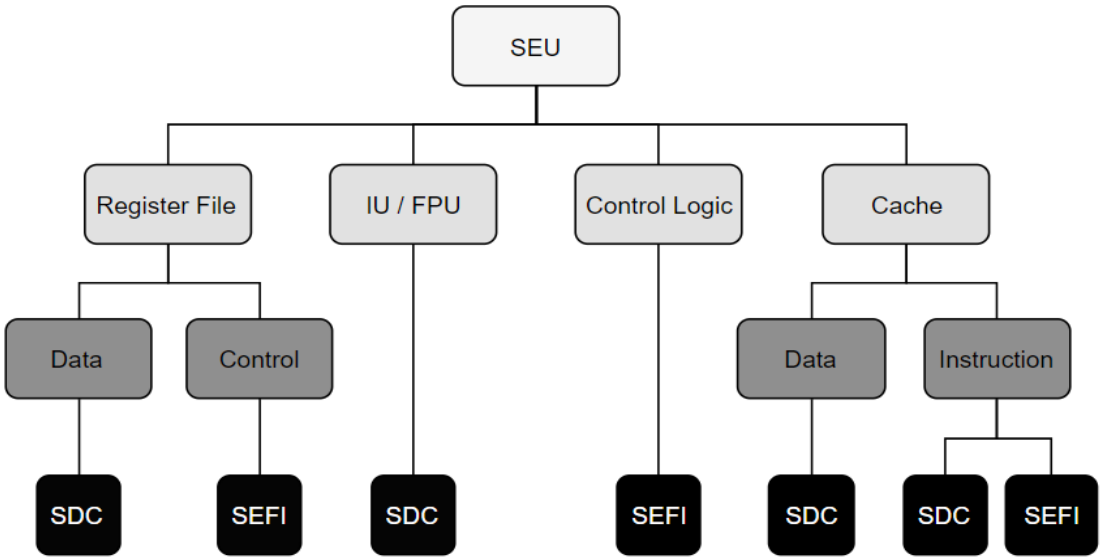
- **Data cache:** These units are split in tag array and data array, like instruction caches. Bit-flips in the tag array may validate out-of-date data, leading to wrong outputs, or invalidate data, introducing a delay in the application (cache miss). If an upset affect the data array, this may lead to corrupted output. However, if the data is out-of-date, the fault is masked, and no effects will be observed.

Fig. 2.2 shows a diagram that presents an overview of the possible effects of SEUs in embedded processors. Concerning soft-core processors, they are even more vulnerable to the radiation effects. The ones embedded in SRAM-based FPGAs are susceptible to persistent soft errors in the configuration bitstream that may affect their functionality.

There are commercial solutions that provide Radiation Hardened (Rad-Hard) processors, which features high tolerance to radiation effects. Such is the case of Microchip (Atmel) Rad-Hard 32 bit SPARC V8 Processor (MICROCHIP, 2017) and VORAGO Rad-Hard ARM® Cortex®-M0 MCU (VORAGO, 2017). These solutions usually integrate Triple Modular Redundancy (TMR) on circuits and registers, and Error Detection and Correction (EDAC) on memories. Although the Rad-Hard architectures provide high reliability, these processors do not achieve performances as high as modern Commercial Off The Shelf (COTS) (AZAMBUJA; KASTENSMIDT; BECKER, 2014). Moreover, other drawbacks include: excessive price; limited use; not widely available - as there are only a few companies in the market; and the fabrication process uses older technologies (GINOSAR, 2012).

The Rad-Hard processors are not the only solution to systems that availability or high reliability are required. The COTS processors can be suitable for such applications if fault-tolerance methods are applied. Further in this chapter, several techniques that can be implemented to improve the dependability of embedded processors are presented.

Figure 2.2: Diagram of soft errors effects in processors



Source: From the author.

2.5 All Programmable System-on-Chip Devices

Heterogeneous computing architectures, which combine embedded processors and FPGAs, are increasingly being used for implementing mission-critical and reliable systems, such as High Performance Computing (HPC) servers, non tripulated vehicles, and avionics systems. In these fields of application SRAM-based FPGA solutions are frequently used due to the high reconfiguration flexibility, competitiveness costs, and capability of integrating complex systems on the same component.

The called All Programmable System on Chips (APSoC) are devices divided in Processing System (PS), which contain dedicated embedded processor, and Programmable Logic (PL), that is the FPGA customizable logic. The embedded processors used in such systems can be based on soft- or hard-cores, in which the former are implemented in the logic elements of the FPGA, and the latter are designed in dedicated silicon. Examples of commercial APSoCs are Zynq-7000 from Xilinx (XILINX, 2016b) and Cyclone-V from Altera (ALTERA, 2015), in which both present dual-core ARM Cortex-A9 processor (ARM, 2010) on the PS and SRAM-based FPGA on the PL part.

The System-on-Chips (SoC) devices that integrate embedded multi-core processors and FPGAs are defined as Multiprocessor System-on-Chips (MPSoC). These SoCs usually contain heterogeneous processors to performance enhancement. Applications with high concurrency can be scheduled to be performed into throughput optimized cores,

and the ones with fewer and complex threads can be programmed to latency optimized cores. For instance, the Xilinx Zynq UltraScale+ MPSoC (XILINX, 2017b) is composed of dual- and quad-core variants of the ARM v8-based Cortex-A53 combined with dual-core ARM Cortex-R5 processor, besides the FPGA part.

Although modern SoCs devices offer a plethora of advantages, they present an additional susceptibility to errors, since the FPGA is significantly affected by radioactive environments (TAMBARA et al., 2016). The ones that use SRAM-based technologies are very susceptible to soft errors caused by radiation effects, as they are composed of millions of SRAM cells used to configure all the synthesized logic, the embedded processors, DSPs, and memories (SIEGLE et al., 2015). Soft errors affect FPGAs by modifying the configuration bitstream, which can cause changes in the logic, functionality, and performance of the device.

2.6 Fault-Tolerance Techniques in Embedded Processors

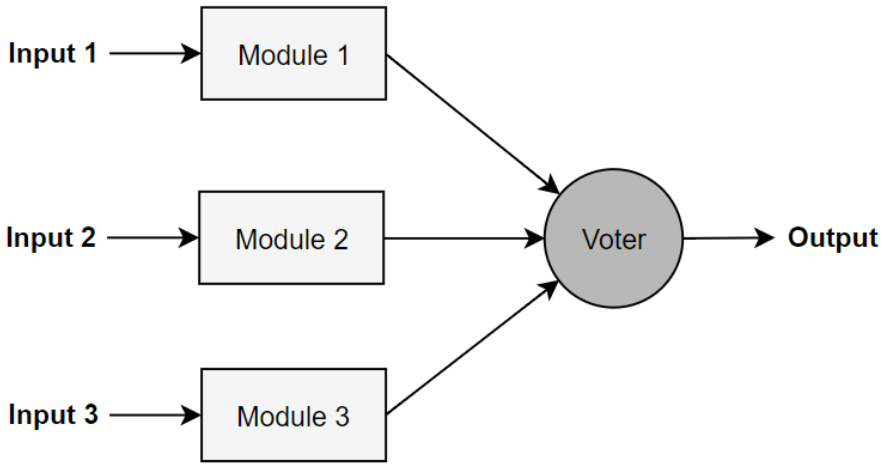
Fault-tolerance techniques to improve the dependability of embedded processors galore are described in the literature. They can be classified as hardware-, software- and hybrid-based techniques (AZAMBUJA; KASTENSMIDT; BECKER, 2014). The next sections present an overview of the main techniques for each classification.

2.6.1 Hardware-based techniques

The Hardware-based techniques are implemented at physical level. Mainly based on redundancy, these methods can provide two or more instances of a hardware component, as processors, memories, buses or power supplies (DUBROVA, 2008). The spatial redundancy techniques are effective for single fault models, in which a soft error affect only one redundant module. If more modules are affected the effectiveness is not guaranteed (ROSSI et al., 2005).

The Duplication With Comparison (DWC) is a technique applied for error detection. As there are only two hardware copies with comparison, the method is not able to recover the system. Thus, when a fault is detected, the system must be restarted to return to a safe state. For error detection and correction, a Triple Modular Redundancy (TMR) can be applied, as presented in Fig. 2.3. Three redundant modules and a voter for com-

Figure 2.3: Diagram of TMR with single Voter.



Source: From the author.

parison are used to mask an error. However, as in DWC, the system must be restarted to guarantee a safe state. In order to avoid single point of failures, the voter can also be triplicated, as shown in Fig. 2.4.

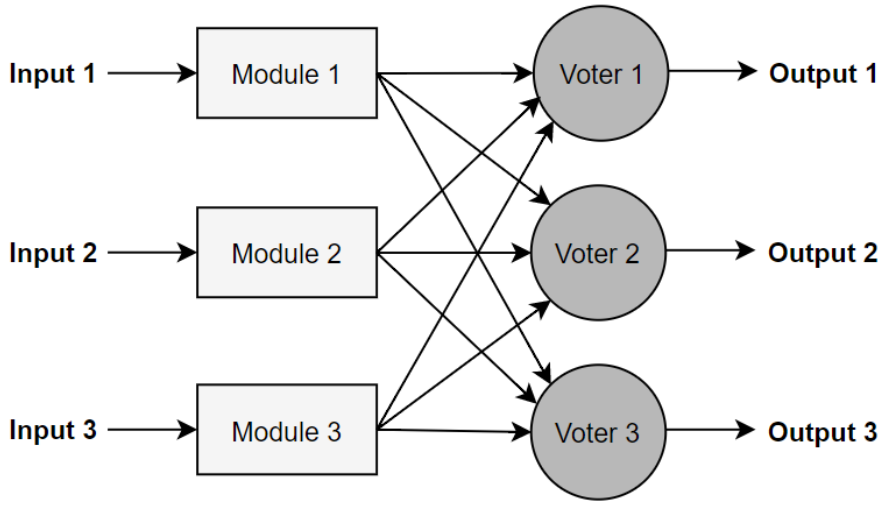
Another type of hardware-based technique is the hardware monitors that can use watchdogs, checkers, or IPs to monitor the system (AZAMBUJA; KASTENSMIDT; BECKER, 2014). The most common, the watchdog processor (MAHMOOD; MCCLUSKEY, 1988) is a module that detects errors by verifying the control-flow and memory access of the target processor. As advantages, less hardware is required compared to replication techniques, and stuck-at errors can be identified. However, the error correction is not supported by this approach.

As previously mentioned, there are commercial processors called Rad-Hard that provide high dependability by applying hardware-based techniques in the internal architecture. Although these methods are efficient, they introduce significant overhead mainly in area and power consumption.

2.6.2 Software-based techniques

The Software-Implemented Hardware Fault Tolerance (SIHFT) (GOLOUBEVA et al., 2006) are techniques that deal with faults affecting the hardware by protecting only the software, without hardware modification. These methods rely on adding redundancy and comparison for error detection.

Figure 2.4: Diagram of TMR with triplicate Voters.



Source: From the author.

The most common types of redundant techniques are the Dual Modular Redundancy (DMR) and the TMR schemes. As mentioned, TMR can also be applied at hardware level to provide fault-tolerance. There is also the N-modular redundancy (NMR), which uses N application replicas. For instance, in multi-core systems, the N replicas can run on different cores. These solutions offer high fault detection (and error masking, in case of three or more redundant executions). However, they introduce a large area, performance, and energy overhead, which can be unacceptable for some applications. Another drawback is these methods only protect the system from SDCs. They are not able to deal with SEFIs that are the most problematic type of error, as it impacts the application execution and the user experience directly.

The redundant-based techniques usually adopt one of these strategies (OSINSKI; LANGER; MOTTOK, 2017): information, temporal or spatial redundancy. The first consists of adding extra information to all the program data, by using Error Detection Code (EDC) or Error Correction Code (ECC) or even by replicating all data in different memory spaces. The temporal redundancy means additional execution time, as the code protection runs in distinct periods during the program execution on the same hardware. For instance, if for NMR the N replies are executed sequentially in the same processor. Another temporal redundancy is applying checkpoint and rollback operations for error recovery. The execution of parallel redundancies in different cores is considered spatial redundancy, likewise the use of additional components.

The presented redundant strategies can be applied at instruction and thread levels to provide protection (OSINSKI; LANGER; MOTTOK, 2017). The approaches that

operate at thread level are called Thread-Level Redundancy (TLR). The concept of applying Simultaneous Multithreading (SMT) to provide fault coverage was first introduced in (REINHARDT; MUKHERJEE, 2000). The authors presented the Simultaneous and Redundantly Threaded (SRT) processor, which runs the program replies simultaneously in independent threads. From this, there was an evolution from single-core to multi-core approaches. The following are examples of single-core implementations: SRT (REINHARDT; MUKHERJEE, 2000); Simultaneously and Redundantly Threaded processors with Recovery (SRTR) (VIJAYKUMAR; POMERANZ; CHENG, 2002); and Software-based Redundant Multi-Threading (SRMT) (WANG et al., 2007). To multi-cores is relevant to cite: Chip-level Redundant Threading (CRT) (MUKHERJEE; KONTZ; REINHARDT, 2002); Chip-level Redundantly Threaded multiprocessor with Recovery (CRTR) (GOMAA et al., 2003); Reunion (SMOLENS et al., 2006); and Dynamic Core Coupling (DCC) (LAFRIEDA et al., 2007).

The power consumption in redundant approaches has been a target of recent researches (SALEHI et al., 2015; SALEHI; EJLALI; AL-HASHIMI, 2016). They study energy efficiency of multi-core platforms that are protected by techniques with redundant solutions. The state of the art of redundancy in multi-core relies on applying the concepts of approximate computing (HAN; ORSHANSKY, 2013) to the N task replies as a way to reduce the power consumption for the reliability improvement (BAHARVAND; MIREMADI, 2017). Although this type of approach provides protection with lower energy overhead, the target application has to deal with inexact results that in most of the safety-critical applications are not acceptable.

The Selective Software-only Error-detection Technique using Assertions (S-SETA) technique (CHIELLE et al., 2015) can deal with control-flow faults that leads to SEFIs. In case of faults, the approach puts the system in a fail-safe state and sighs the fault occurrence, allowing the designer to apply a recovery solution. Thus, this technique can detect SEFI by control-flow analysis but does not correct the errors. However, SETA can be combined with data-flow techniques to increase the reliability.

Considering the SEU susceptibility of embedded OSes, Rodrigues and Kastensmidt (2017) evaluated the behavior of approximative applications running in bare-metal and on top of FreeRTOS and Linux. The authors have concluded that the FreeRTOS has less impact on the system susceptibility to errors than the Linux. However, the FreeRTOS applications are much more prone to hangs than the Linux and bare metal counterparts. Besides, for Linux executions, the segmentation fault errors are an issue, which is not

present on FreeRTOS and bare-metal. In (FAYYAZ; VLADIMIROVA, 2014), an approach is proposed on fault tolerance for aerospatial FreeRTOS applications. The authors have developed a technique that makes use of parallelism to provide safety. The method consists of migrating tasks from a faulty processor to a healthy one. The problem with this approach is that a high amount of assumptions must be taken beforehand by the designer. Decisions like which tasks will be migrated to which nodes are made in the design phase. Unfortunately, a programmer is not able to safely predict which processing nodes will fail. Thus, a more general method to protect the system should be used that requires no previous assumptions to be made by the designer. As further presented, the lockstep technique can be applied to such systems, providing a high reliability.

2.6.3 Hybrid-based techniques

The hybrid techniques are the ones that uses a SIHFT method combined with a hardware module that performs consistency checks in the processor. The work in (CHIELLE et al., 2016) combines a set of SIHFT techniques with a Control Flow Checking (CFC) module to monitor the trace port of the processor. Whereas Azambuja et al. (2013) presents a Hybrid Error-detection Technique using Assertions (HETA), which is a hybrid method that uses a watchdog module and assertions (or signatures) to deal with control-flow errors.

Lockstep is a hybrid fault-tolerance technique based on software and hardware redundancy for error detection and correction (ABATE; STERPONE; VIOLANTE, 2008; VIOLANTE et al., 2011; GOMEZ-CORNEJO et al., 2013; PHAM; PILLEMENT; PIESTRAK, 2013). It uses the concepts of checkpoint combined with rollback mechanism at software level, and processor duplication and checker circuits at hardware level. A typical lockstep works by executing the same application simultaneously and symmetrically in two identical processors, which are initialized to the same state with identical inputs (code, bus operations, and asynchronous events) during system start-up. Therefore, during normal operation, the state of both processors is identical from clock to clock. In an error-free execution, they are expected to perform the same operations allowing the monitoring of the processor's data, addressing, and controlling buses (BOWEN; PRADHAM, 1993). There is a checker module that monitors the CPUs and periodically compares the outputs to check for inconsistencies. Points of verification must be inserted in the program to indicate when the application execution must be locked and the outputs compared. If

there is any discrepancy in the results, the lockstep system restores the processors to a safe state through a rollback mechanism. Otherwise, the processors are assumed to be fault-free and a checkpoint operation is performed.

The checkpoint consists in an operation that stores the processor's context in a secure memory, without errors. Memories can be protected by ECC to avoid memory data corruption. ECC can detect and correct single-bit errors, and detect double-bit errors. The processor's context is defined as all data resources used in the application execution - i.e., registers, main memory - necessary to repair the system in case of errors. Rollback is a recovery method that restores the fault-free copy of the processor's context from memory. The processor then is recovered to a state without errors and restarts the application execution from this point.

2.6.4 Summary

An overview of the techniques classification is presented in Table 2.1. The Hardware-based consists of modifying the system's architecture, adding checker modules and physical redundancy (e.g., processors or memory replication), which usually present a high area overhead. The software solutions introduce extra instructions and redundant information (e.g., replication of the entire program or parts of it) to be able to detect or correct soft errors in the processor. This type of approach usually presents a high overheard in the execution time. Finally, the hybrid ones integrate the hardware- with software-based techniques in order to improve error detection and correction and to reduce area and time overhead.

The main drawback of the presented methods is they only protect the system from SDCs or SEFIs, individually, hardly both at the same time. Most of the described fault-tolerant solutions based on redundancy are used to protect the processor against soft errors that lead to SDCs, but no SEFI. Besides, the results verifications are mostly performed by the own processors, which could increase the errors susceptibility and also false detections if a fault affects the checker in the processor. Aiming to solve these topics a straightforward solution is to use an external module that compares the outputs and also controls the processor operation as a way to detect a system crash and perform some recovery mechanism. However, once heterogeneous architectures are considered, the techniques to monitor and observe fault effects, track errors and perform recomputation are more challenging. In APSoCs, the system verification could be handled by a module centralized

Table 2.1: Overview of the techniques classification

Technique Classification	Pros	Cons
Hardware	- High fault detection - Fast detection - No software modification	- Most does not correct errors - Mainly single fault model - High area and power overhead - Implemented only in physical level - Can be expensive
Software	- High fault detection - High flexibility - No hardware modification - Small area overhead - Some can correct errors	- High performance overhead - Mainly single fault model - Focuses only on data or control flow, but not both
Hybrid	- High fault detection - High efficiency - Can achieve small area overhead - Some can detect both SDC and SEFI - Some can correct errors	- Can also achieve high performance or area overhead - Software and hardware modification

or distributed, and the hybrid lockstep technique could be applied. The method implemented in this work is based on lockstep and is capable of detecting both SDC and SEFI errors before they lead to catastrophic issues. The next section presents an overview of the lockstep related works.

2.7 Related Works about Lockstep Technique

There are processors, like ARM Cortex-R5 (ARM, 2011), that already provides in the architecture support to lockstep mode, which can be configured to application reliability. The difference is the implemented technique is only hardware-based. The same program runs on distinct lockstep cores, which allows error detection, for DMR version, and error correction, for TMR version. However, for those processors that do not have this functionality, the lockstep has to be implemented. Several approaches of lockstep have been described in the literature to improve the system dependability in applications that require high reliability.

The idea to combine the rollback recovery with checkpoints for hardening processor cores inside FPGAs was introduced by Abate, Sterpone and Violante (2008). The authors presented a lockstep architecture adopting a Virtex II-Pro target device which em-

beds two hard-core PowerPC processors. The memory to save the context is assumed to be immune to SEUs, which is an unrealistic scenario and the memory susceptibility also have to be considered. By fault injection in the processor's registers (user, special purpose, and control registers), the authors concluded that the implemented approach was able to correct up to 54% of the injected faults and 3% led to wrong results. The processor verification is performed in every write cycle. Thus, all the information sending to the memory or the peripheral devices is sanity checked. Although this is an efficient verification method, this may lead to a huge performance overhead. Unfortunately, the authors did not mention the implementing cost.

The lockstep approach presented in (VIOLANTE et al., 2011) aims to protect a soft-core processor Leon2 implemented into a Xilinx Virtex. The work is an extension from (REORDA et al., 2009). The authors take advantage of the register window mechanism - a feature of the Leon processor - to apply the checkpoint and rollback in efficient ways. Besides, the fault injection attacks the pipeline registers, which originates a deterministic behavior with predictable results. Thus, the checker module implementation can be adjusted to specific cases. Every memory access (both write and read cycles) is analyzed to verify the processor consistency. The execution time overhead reported ranges from 17% to 54% depending on the amount of data. The experimental results show that the technique could detect and correct 20% of soft errors injected in the processor's registers and 79% of the faults were effectless. The authors irradiated the system for 24 hours and observed 254 SEUs that resulted in rollback recovery and 13 SEUs led in device reconfiguration.

The solution from (GOMEZ-CORNEJO et al., 2013) applies lockstep to an adapted 8-bit soft-core processor based on the PIC16 architecture. The design was implemented in a Xilinx Virtex-5 FPGA. The architecture is composed of two processors, a lockstep controller, four BRAMs memories (one for data; one for instructions; and two for context backups), a DMR Comparator, and other IPs used to ECC and data mixer. The area overhead is around 300%. To minimize the context saving and recovery latency, the authors made several adaptations in the processor architecture. Consequently, this also leads to a high fault recovery rate in the simulated experiments. For the fault injection emulated in the board, upsets are injected in the registers of only one processor by connecting some of the registers' bits to an IP that generates error signals. However, it is not clear the target registers and how many bits can be affected. To evaluate the correctness of the system a serial transmission application sends to the computer a sequence of ASCII characters

stored in the program memory. For these experiments, none of the injected faults affected the received data on the computer. The authors did not present numbers showing how many upsets were corrected by the implemented approach.

Pham, Pillement and Piestrak (2013) proposed an enhanced lockstep scheme using two soft-core MicroBlaze processors implemented in a Xilinx Virtex-5 FPGA. The novelty of the work is the possibility of identifying the faulty core in case of error through the use of a Configuration Engine, which is built using PicoBlaze cores. Differently from most of the lockstep approaches, checkpoint and rollback mechanisms are not used to recover the system from an upset. Instead, they use a combination of partial reconfiguration with roll-forward recovery technique. The lockstep scheme area overhead is around 297% against 384% using the TMR approach, which is also evaluated in the work. A fault injection in the configuration bits revealed that 8.6% bits of the MicroBlaze core are sensitive, where 2.3% cause persistent errors. The average time duration to recover the processor in the enhanced lockstep is 23us, meanwhile in the basic lockstep is about 516us.

The lockstep technique is extensively used in semiconductors companies for device enhancement. For instance, Texas Instrument proposes a delayed lockstep in a dual-core architecture microcontroller (TROPPMANN; FUESSL, 2008). Different delay stages are applied to the CPUs in lockstep in order to detect errors. Another example is Freescale Semiconductor Inc. that in (MOYER; ROCHFORD; SANTO, 2012) presents a multi-core data processing system where the processors operate in lockstep. In this architecture, errors in the lockstep synchronization are prevented by forcing the cores to be in the same state even when an internally generated exception is provoked by soft errors.

The main drawback of the lockstep technique is related to the context saving and recovery process latency. For real-time applications, performance penalties are critical. Nevertheless, for some applications, the overhead is acceptable given the improvement of the system reliability.

The majority of lockstep approaches are implemented to protect soft-core processors (VIOLANTE et al., 2011; GOMEZ-CORNEJO et al., 2013; PHAM; PILLEMENT; PIESTRAK, 2013). As main advantage, these cores have an open architecture, which the designer can modify if needed. On the other hand, soft-cores embedded into SRAM-based FPGAs are susceptible to persistent bit-flips in the configuration bitstream. Therefore, periodically reconfiguration (scrubbing) must be performed to correct the persistent errors. The approach implemented in this work focuses in hard-core processors embedded into

APSoCs. Thus, a high-performance processor is used, also taking advantage of the programmable logic. Therefore, persistent errors in the embedded processors caused by soft errors in the programmable matrix are avoided, differently as occurs with soft-core processors. Differently from others lockstep approaches that attach both processors to the same memory (ABATE; STERPONE; VIOLANTE, 2008; VIOLANTE et al., 2011), in this work each CPU is connected to its own private BRAM memory. Thus, single point of failures in the memory data is avoided. Concurrency to shared memory resources is also minimized. Another important aspect of this work is the possibility of the processors work with distinct clocks, differently from a typical lockstep approach that both CPUs must be in the same state clock to clock. Moreover, this work applies the lockstep approach to applications running on top of FreeRTOS, which is an OS extensively used in industry. A comparison of bare-metal and FreeRTOS fault mitigation performance is also presented, which is not shown on previous works of this kind. Finally, a complete evaluation of the impact in used the proposed lockstep in the system is presented, none of the past works made this analysis.

3 PROPOSED DUAL-CORE LOCKSTEP

This work explores Dual-Core LockStep (DCLS) as a fault tolerance technique to mitigate radiation induced faults affecting hard-core processors embedded into APSoCs. The DCLS architectures are based on redundancy to detect and correct errors that affect the processor operation.

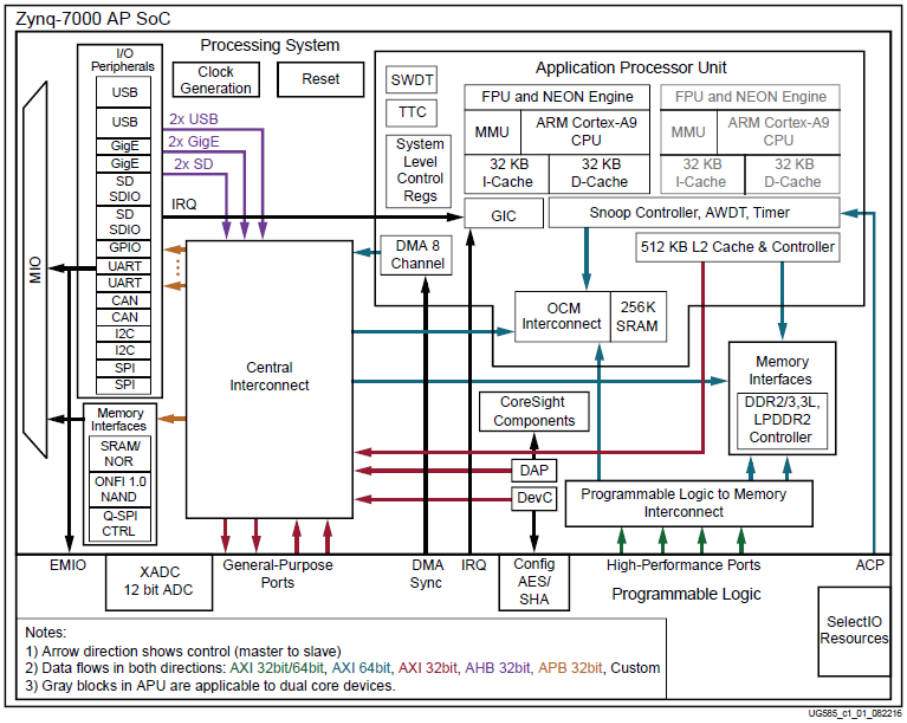
In a typical DCLS system, there are two identical processors executing the same application, simultaneously. Both CPUs run the application symmetrically, allowing monitoring the processors step by step. The DCLS technique assumes that a fault in either processor will cause a difference between their states, which will eventually be manifested as a discrepancy in the outputs. Thus, the outputs of both CPUs are periodically compared by a checker module for error detection. The operations of checkpoint and rollback are combined for error correction, restoring the system to a safe state. This chapter details the implementation of the proposed DCLS applied to protect the embedded processor.

3.1 Case Study

The proposed DCLS technique was designed and implemented on the dual-core ARM Cortex-A9 processor embedded into Zynq-7000 APSoC (XILINX, 2016b) from Xilinx. This device combines a 28nm Programmable Logic (PL) layer with an embedded ARM processor on Processing System (PS), as described on Fig. 3.1. This work uses a Zedboard (AVNET, 2017) featuring an XC7Z020-CLG484 device, which is the Device Under Test (DUT). The system package was thinned to allow the penetration of ions into the active silicon region for the radiation experiments further presented.

The PL part of the DUT presents a 7-series SRAM-based FPGA with 4.9 Mb total Block RAM (BRAM), 85 K logic cells, and 220 DSP slices, with a frequency of 100MHz. The PS is composed of a 32-bit dual-core ARM processor running at 666 MHz, a 256 KB dual-port SRAM On-Chip Memory (OCM) (available for both the CPU and the PL), and several I/O peripherals and interfaces. Each CPU has an independent 32 KB level 1 (L1) cache for data and instruction. A 512 KB level 2 (L2) cache is shared between both processors. The board also counts with an external 512 MB DDR memory. Although the proposed DCLS was implemented to the protect the ARM processor into Zynq, the approach can be extended to different hard processors embedded into APSoCs with a few adjustments.

Figure 3.1: Zynq-7000 APSoC Overview



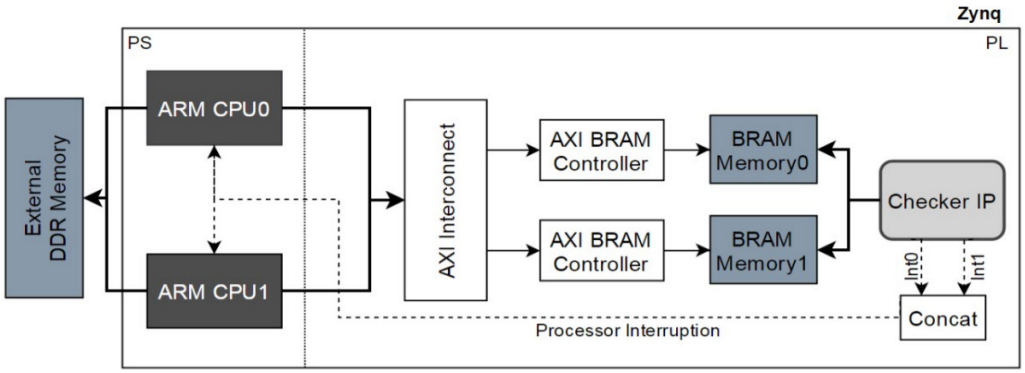
Source: (XILINX, 2016b).

3.2 Architecture

The DCLS system architecture is detailed in Fig. 3.2, which is composed of a dual-core ARM (CPU0 and CPU1); two BRAM memories; an external DDR memory; and a Checker module. In order to avoid single point of failures in the data memory and to minimize both ARM CPUs to contending to shared memory resources, each processor is connected to its own private dual-port 64KB BRAM memory. All the application data and the processor's context are stored in the BRAMs, and its size strongly depends on the application, which shall be resized if needed. As the BRAMs are located in the PL part of Zynq, the access to them is allowed through an AXI interconnect interface. Besides, there is an AXI BRAM Controller responsible for controlling the communication.

The processors are also connected with an external shared DDR memory, which stores both CPUs' program instructions. Although both processors run the same application, the program of each one is saved in distinct addresses of the DDR memory. Additionally, the DDR is used as an alternative safe memory to store the checkpoint of the system, as explained later. Finally, the Checker is a module responsible for verifying the processors consistency. It is connected with both BRAMs memories to access the CPUs' outputs. Besides, there is a Concat used for concatenating the interrupt signals.

Figure 3.2: Proposed lockstep architecture for dual-core ARM Cortex-A9 embedded into Zynq-7000 APSoC.



Source: (OLIVEIRA et al., 2017).

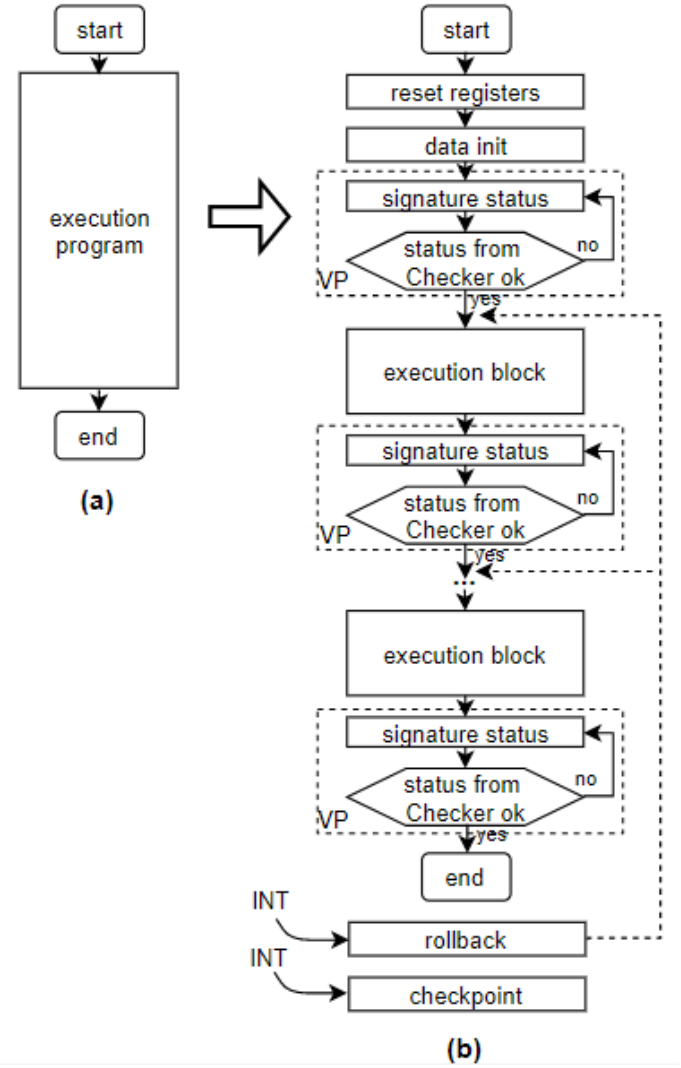
Considering the shared resources, the processors work in handshake mode to execute the application, where the CPU0 is the priority core. Although the processor provides two embedded cache levels (L1 and L2), according to (TAMBARA et al., 2016), the use of any cache memory affects the cross section of the ARM processor, which can highly compromise the system reliability. Thus, all caches are disabled in this work.

3.3 Implementation

The DCLS functional flow for each ARM CPU is described in Fig. 3.3. The unprotected program behavior without lockstep is illustrated in (a). Whereas, (b) represents the steps required for the implemented approach. The original application is divided into execution blocks, and between each one there is a Verification Point (VP). There is also a VP at the beginning and at the end of the execution. Thus, the number of VPs is equal to the number of execution blocks plus one.

Owing to the fact that after the reset the ARM registers are undefined state and they should be compared further, the processor's registers are cleaned at the beginning of the program execution. After the data initialization, the first VP is executed. At this point, the application execution is stalled. The processor status, which is a signature that represent the actual CPU state, is written on the respectively BRAM memory and the processor remains locked waiting for the Checker approval. Then, the processor's context is accessed by the Checker.

Figure 3.3: Lockstep Functional Flow for ARM Cortex-A9 dual-core: (a) original code, (b) code with lockstep technique running in both CPUs



Source: (OLIVEIRA; TAMBARA; KASTENSMIDT, 2017b).

For the sake of this work, the processor's context is assumed as the following ARM CPU's registers:

- General-purpose: R0 to R12;
- Specifics: Stack Pointer (SP); Link Register (LR); and Program Counter (PC).

The interrupt mechanism is used to access the processor's context, as the interruption naturally stores the registers values on the stack, making simple their management. When an interruption is processed by the CPU, the following actions are performed (TAYLOR, 2014):

1. The execution of the actual thread is stopped.
2. The processor's registers are saved into the stack.

3. The interruption routine is executed.
4. At the end of the interruption routine, the processor restores its context from the stack.
5. Continues the execution of previous thread.

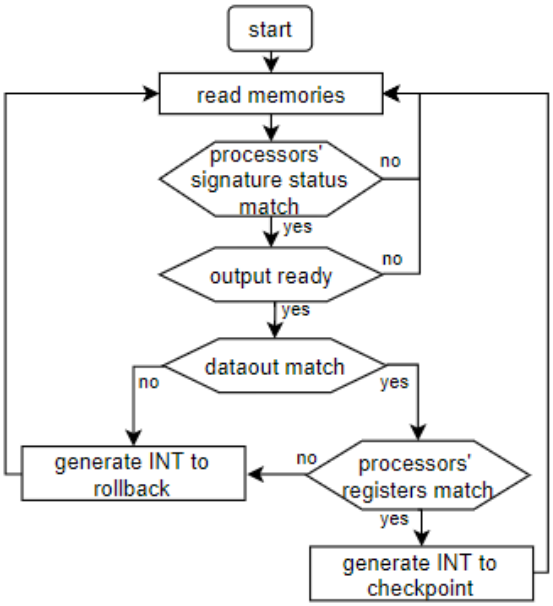
Therefore, at the VP, the Checker generates an interruption to access the processor's registers. Then, the Checker compares the outputs and the register file of both CPUs to find discrepancies. If the results match, it is assumed that the processors do not have any errors and the current system state is consistent, which can be saved for future reuse. Thereupon, a new interruption is launched to each CPU to perform a checkpoint operation. If any discrepancy between processors data is found, which implies in the occurrence of errors, an interruption is generated to perform the rollback method, recovering the system. At the end, the Checker writes a status on both BRAM memories to unlock the processors. Once unlocked, the CPUs continue running the application until reach the next verification point, and the cycle is repeated.

3.3.1 Checker Module

The Checker module is a customized Intellectual Property (IP), designed in Hardware Description Language (HDL) and implemented as a Finite-State Machine (FSM) in the PL of Zynq. It is responsible for verifying the processors consistency and controlling the lockstep approach. To identify if there is an error in the system, the Checker compares the outputs of both processors. This comparison can be performed in two ways: accessing all the output data stored in the BRAM memories and comparing each position, or previous applying a signature to the outputs and the Checker compares just the signatures. In the first case, there is no additional code on the application and, on the verification point, the Checker verifies all the defined outputs positions from both BRAM0 and BRAM1. In the second case, a signature must be calculated on the application before the verification point. Thus, the Checker compares just the position of the signature on the memories. This signature could be implemented straightforwardly as a sum of all elements, a xor mask, a typical checksum or even another method that summarizes the processor's results.

Fig. 3.4 presents an overview of the Checker behavior. First, it read the BRAMs memories aiming access both processors status, which means the current CPUs state. If

Figure 3.4: Checker module functional flow



Source: (OLIVEIRA; TAMBARA; KASTENSMIDT, 2017b).

the status indicates outputs ready, the Checker compares the data of both processors. To make sure that the processors are in a correct state, besides the outputs verification, the Checker compares the registers. If there is any mismatch in the results (outputs or register file), an interruption is generated to both ARM CPUs indicating a rollback operation. Otherwise, an interruption to perform a checkpoint is launched.

Regarding the verification of the processors' registers, some characteristics are required to be considered:

1. At the beginning of the program execution, the general-purpose registers have to be clean.
2. Each CPU is connected to independent BRAM memories implemented in FPGA logic and mapped to different addresses.
3. Although both CPUs run the same application, the program instructions of each one are stored in distinct addresses of the DDR memory.
4. The architecture of the ARM processor (ARM, 2012) defines that general-purpose registers (R0 to R12) can be used by the software when the CPU is in user mode, but there are system modes that could access some of these registers to store instructions or system information.
5. ARM deprecates the use of the specifics registers (SP, LR, and PC) for any purpose other than as they are specified for; the incorrect handling of these registers could

lead the system to unpredictable behavior.

Concerning all the presented points, the R0, R1, R11, R12, SP, LR and PC registers could not be protected by the proposed solution, because these registers have distinct values in the processors during a fault-free execution.

Besides the SDC detection, the Checker also verifies processor crashes. A watchdog timer is configured to twice the time required to run an application block. If a CPU did not reach the verification point before the watchdog timer is over, it is considered a system inconsistency and the Checker operates the rollback mechanism. Thus, the system's dependability is guaranteed.

3.3.2 Checkpoint and Rollback Methodology

A checkpoint is an operation that saves a consistent state of the processor in the memory, and the rollback recovers the system from an error by restoring that previous state. When the Checker verifies that the system is fault-free, it generates an interruption request to each CPU, allowing to access and to save their context. Thus, when the *checkpoint interrupt routine* is executed it accesses the processor's stack and makes a copy of the registers to the memory. On the other hand, when the Checker detects a mismatch in the CPU's data output, the interrupt is launched to perform a rollback. In the *rollback interrupt routine* the processor's stack is overwritten with the registers stored in the memory. When the processor restores the context from the stack, the system returns to a consistent state.

In this work, there are two designs explored to perform a checkpoint and rollback, as described below:

- **DCLS accessing BRAM memory only (DCLS_BR)**: The processor's context is saved only in the respective BRAM memory. Besides, the BRAMs are responsible for storing the application data, as explained before. When a rollback is performed, the processors' registers are overwritten with the corresponding values from the BRAMs.
- **DCLS accessing BRAM and DDR memories (DCLS_BR_DDR)**: At the checkpoint, the context is stored in both the BRAMs and the external DDR memory. Furthermore, all the application data saved in the BRAMs are copied to the DDR. Although both memories present EDAC capability, MBUs can occur, affecting the

stored data. The duplication is done to increase data reliability further. In the rollback operation, the context is read from the BRAMs. If after the block re-execution a mismatch is still present in any results (outputs or register file), a new rollback is performed, this time using the data stored in the DDR. In this case, all information previously saved on the BRAMs are overwritten by the DDR data.

To deal with errors that could occur between the verification point and the context storage the first checkpoint is saved in two different BRAM memory addresses. In which the first is overwritten in every checkpoint and the other still unchanged. If two consecutive rollbacks are performed, without context storage between them, this indicates that the actual checkpoint has an error. Therefore, the system is recovered to the first context saved, returning to the beginning of the application. Although this approach has a performance penalty, it avoids a hang in the system caused by infinite rollback to a wrong context.

Four levels of rollbacks are implemented to recover the system. If a rollback is performed and the system still presents an error, the next rollback level of the sequence is executed. The levels are described below:

1. From BRAMs: The first rollback level read the context from the BRAMs memories to recover the system.
2. From DDR: Only used in the DCLS_BR_DDR design. This rollback restores the context from the DDR memory.
3. First context: After trying to recover the system with the most recent context saved, without success, the first context is used.
4. Application reset: If the others levels do not have effect or if there is not even one checkpoint, the application is re-started to recover the system.

If after applying all the rollback levels the DCLS does not recover the processor successfully, this mean that a permanent fault or crash occurs. Thus, a soft reset is executed. This reset is performed through the control of the processor's private watchdog (XILINX, 2016b).

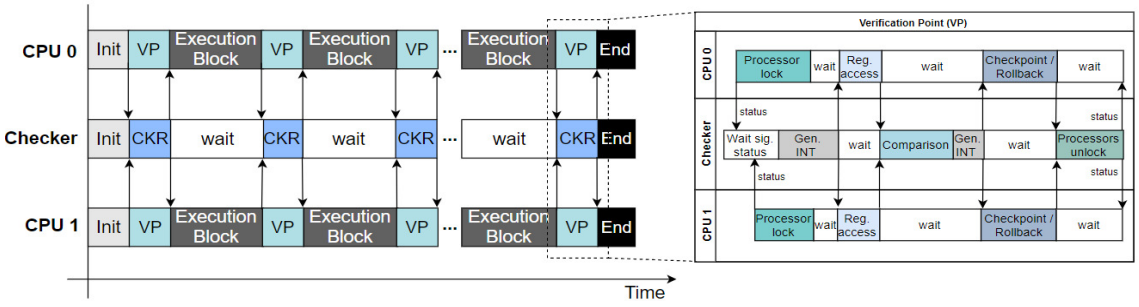
3.4 DCLS approach overview

The DCLS approach works by executing the same application, partitioned in blocks, in both ARM cores in parallel. Fig. 3.5 details the execution overview of both CPUs and the Checker. Besides, there is the sequence flow of the verification point. As described in 3.3, there are VPs in the beginning, in the end, and between the execution blocks.

When the execution reaches a verification point, the processor status is written to memory, and the execution is locked. After both CPUs status is written, the Checker generates an interruption to access the processor's registers. Thus, all the results (outputs and registers) of both CPUs are compared by the Checker. If no difference is found, the system is considered to be in a safe state, and another interruption is generated to each CPU to operate a checkpoint. If any discrepancy between processors data is found, an interruption is launched to recover the processors using the rollback mechanism. At the end, the Checker writes a status on both BRAM memories to unlock the processors. This cycle goes on until the end of the application.

The DCLS implementation is also detailed in the works (OLIVEIRA; TAMBARA; KASTENSMIDT, 2017a) and (OLIVEIRA; TAMBARA; KASTENSMIDT, 2017b).

Figure 3.5: Dual-Core Lockstep execution overview



Source: From the author.

4 EXPERIMENTAL METHODOLOGY

The DCLS approach shall be evaluated on fault detection and mitigation. For that propose, experiments that emulate realistic aerospace environment are required. Aiming to validate the system dependability and behavior under faults, the proposed DCLS was submitted to fault injection and radiation experiments.

The fault injection experiments are used to achieve detailed assessment of the DUT. As this tests provide a high control of the affected area, the system vulnerability and sensitivity can be appropriately analyzed. In the literature, there are several methods implemented to emulate faults in the system. Some approaches do FPGA emulation, using reconfiguration mechanisms (NAZAR; CARRO, 2012) or hardware prototyping (ENTRENA et al., 2012), to inject faults at Resistor-Transistor Logic (RTL) level. Others inject faults at instruction level (MANIATAKOS et al., 2011) or in the processor's registers (RODRIGUES et al., 2017) by simulation, which can be a more accurate injection. Besides, there are the software-based techniques that can access sensitive areas of the processor (as registers or memory) and inject bit-flips (VELAZCO; REZGUI; ECOFFET, 2000; LINS et al., 2017).

The closest to real radiation scenarios are radiation experiments, which expose the system to an accelerated flux of particles. The target DUT is exposed to energized particles that emulate the ones present in space. Heavy ions, protons, and neutrons can be accelerated by a facility and thrown at the DUT package. In this case, the entire system is affected by the radiation effects, making impossible identify the total affected area. There are just a few radiation facilities around the world. As examples, the Los Alamos National Laboratory (LANL) in the United States and Laboratório Aberto de Física Nuclear at Universidade de São Paulo (LAFN-USP) in the Brazil conduce neutrons and heavy ions tests, respectively. The main drawbacks of such experiments are the complexity, cost and time to perform it.

For a realistic evaluation of the system, the radiation experiments are required. On the other hand, the fault injection provides a better assessment than radiation. Therefore, the efficiency of the proposed DCLS is validated in both fault injection and radiation with the tests performed directly into Zynq. This chapter describes the methodology used to reach the results. The following sections describe the procedure to perform the experiments, the evaluated applications and the software optimizations applied to assess the system.

4.1 Fault Injection Experiments

A fault injection method was implemented to emulate faults in the dual-core ARM processor (OLIVEIRA; TAMBARA; KASTENSMIDT, 2017b). Bit-flips are randomly injected at the register file to simulate soft errors. The susceptible registers to upsets from the fault injection are the same used in the processor's context:

- General-purpose registers: R0 to R12;
- Specifics registers: SP; LR; and PC.

The injector is desired to be less intrusive as possible to the evaluated system. Past works achieve fault injection with low algorithm intrusion (VELAZCO; REZGUI; ECOFFET, 2000; LINS et al., 2017), making use of interruption mechanisms. This work also uses this strategy. Moreover, as the target of the fault injection is the processor's registers, the interrupt mechanism is a straightforward way to access them.

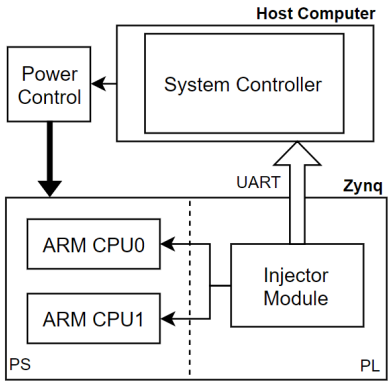
The architecture of the fault injection method implemented is detailed in Fig. 4.1. The system is composed of the following modules:

- **Power Control:** An electrical device responsible for powering up the board in each injection cycle.
- **System Controller:** Software application located in a host computer that is in charge of managing the Power Control and storing the fault injection logs received by serial communication.
- **Injector Module:** A customized IP, implemented in the FPGA layer of Zynq and designed in HDL, responsible for performing the fault injection procedure.

Fig. 4.2 shows the flow diagram of the fault injection proceeding. In the first step, the Injector is configured with a random injection data, which contains the injection time and the fault target location. The latter is defined as the CPU and register in which the fault must be injected, besides the specific bit to be flipped. Due to the complexity of generating random numbers in FPGA logic, the injection configuration is rendered by the CPU0 before it starts the application and, in sequence, the Injector Module is set up. The injection time is defined based on the execution time of the application, which means that a fault could be inserted at any moment during the application time, as in real scenarios.

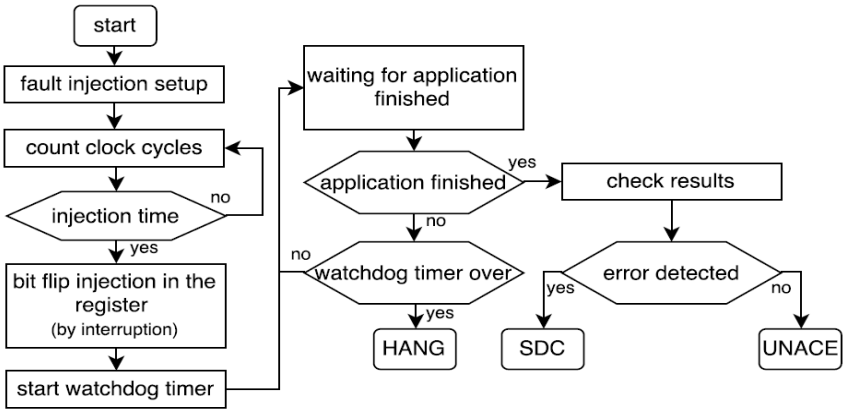
Once the Injector Module has been configured, it starts to count clock cycles until it reaches the injection time. Then, the injector launches an interruption to the specific CPU indicated by the configuration. In the *injection interrupt routine*, the target register

Figure 4.1: Fault injection experiment setup



Source: From the author.

Figure 4.2: Fault injection procedure flow



Source: (OLIVEIRA; TAMBARA; KASTENSMIDT, 2017b).

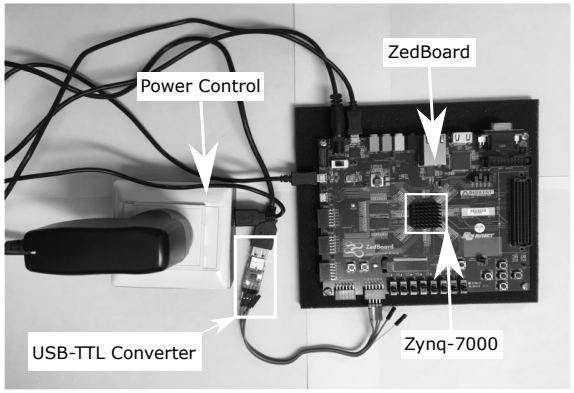
is read from the stack, an XOR mask with the appropriated bit to flip is applied to its value and, then, the register is overwritten.

After injecting a fault, the Injector Module starts a watchdog timer with twice the value of the application time and remains to wait for the end of the application. If the application does not finish before the watchdog timer is over, it is considered an occurrence of a Hang, which is defined as a crash in the system or an infinite loop in the application. In case of the application finished on time, the Injector Module verifies the results generated by both CPUs with the gold ones. If there is any mismatch, it is indicated that an SDC occurred. The fault is classified as UNACE when the injected bit-flip does not affect the system. Two others classifications are made for the DCLS design analysis: Masked Faults and Mitigated Faults. The former occurs when a fault is detected only in the register file, and it is successfully masked. In this case, the fault is corrected before lead to an output error. The Mitigated Faults represent the errors or crashes detected and correctly recovered by a rollback in the DCLS approach. This classification also represents the

Table 4.1: Error classification description

Error Classification	Description
UNACE	Ineffective faults
Masked Faults	Masked silent faults
Mitigated Faults	Corrected by rollback or soft reset
SDC	Output errors
Hang	System crashes

Figure 4.3: View of fault injection experiment setup



Source: From the author.

faults that are mitigated by a soft reset. Table 4.1 summarizes the error classification.

Fig. 4.3 shows the fault injection experiment environment. The ZedBoard and the Power Control are connected to the host computer, where is running the System Controller. Besides, the USB-TTL Converter, responsible for transmitting the serial data, is attached to the board and the computer.

4.2 Radiation Experiments

The Experiments were conducted through heavy ions tests performed using the 8UD Pelletron accelerator at Laboratório Aberto de Física Nuclear at Universidade de São Paulo (LAFN-USP), Brazil (AGUIAR et al., 2014). The used beam line (MEDINA et al., 2016) achieves the requirements of European Space Agency (ESA) (ESA, 2005) for radiation tests in electronic devices. The heavy ion beam has a high uniformity, low intensity flux, and large area. These can be produced by scattering in a gold foil and defocusing technique (MEDINA et al., 2016).

In order to observe the SEU events, the DUT was irradiated in-vacuum using a 39 MeV ^{16}O beam at $90°$. This beam produces a Linear Energy Transfer (LET) of about 5.5 $MeV/mg/cm^2$ on the active region and penetration of around 24 μm in Si. The beam flux has a uniformity of around 93% in 4.0 cm^2 of beam area. For the experiments, the regular beam flux was from 2.19×10^2 to 4.07×10^2 $particles/cm^2/s$. By static tests, these fluxes produce an average of 5 $bit\text{-}flips/s$ in the PL part of Zynq. The beam flux was experimentally defined to achieve the desired error rate without permanent damage to the DUT. Previous works used similar beam flux (TAMBARA et al., 2016; TAMBARA et al., 2015). Besides, this low flux is following the recommended by ESA for SEU tests: flux ranging from 10^2 to 10^5 $particles/cm^2/s$ (ESA, 2005).

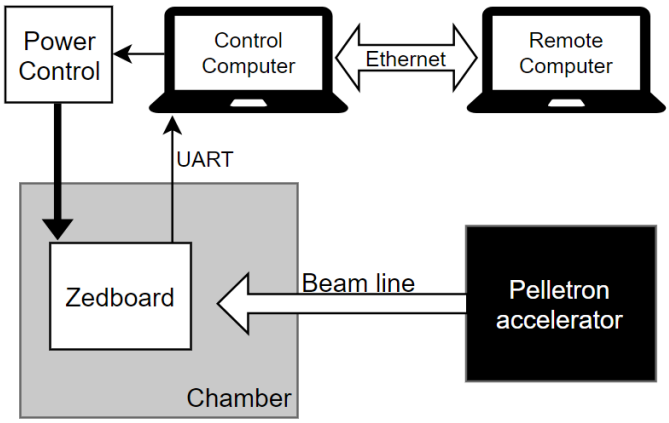
Fig. 4.4 details the experiment setup performed at LAFN-USP. The beam flux affects just the Zynq XC7Z020-CLG484 package that was thinned to ensure the penetration of ions in the active silicon region. The Zedboard, which is located inside of a vacuum chamber, is connected to a control computer and the Power Control (same used in the fault injection tests). Because the environment is not safe for humans being during the experiments, there is a remote computer communicating with the control computer. The perspective of the mounted setup is present in Fig. 4.5, where (a) shows the view inside of the chamber and (b) outlooks the laboratory.

As the goal of the experiments is mainly to observe the soft errors affecting the ARM processor, techniques were applied to reduce the SEFIs in the PL part. The 7 series FPGA devices from Xilinx (XILINX, 2016a) includes a feature that combines continuous readback of configuration data with Frame ECC for SEU detection and correction. Besides enable this feature, a TMR was implemented to protect the Checker IP. Although these techniques are useful, they are not enough to prevent that accumulated faults in the DUT compromise the experiments. Thus, a periodical reset was configured to avoid that bit-flips accumulate over time in the PL and the processor. Every one minute the board was restarted. In that period, the average of accumulated bit-flips in the PL is around 300.

The cross section is used to evaluate the tested designs' susceptibility to soft errors. This standard metric represents the radiation-sensitive area of the DUT by the relation between the number of errors and the total particles fluence, as defined in Eq. 4.1.

$$CrossSection = \frac{\#Errors}{Fluence} \tag{4.1}$$

Figure 4.4: Radiation experiment setup

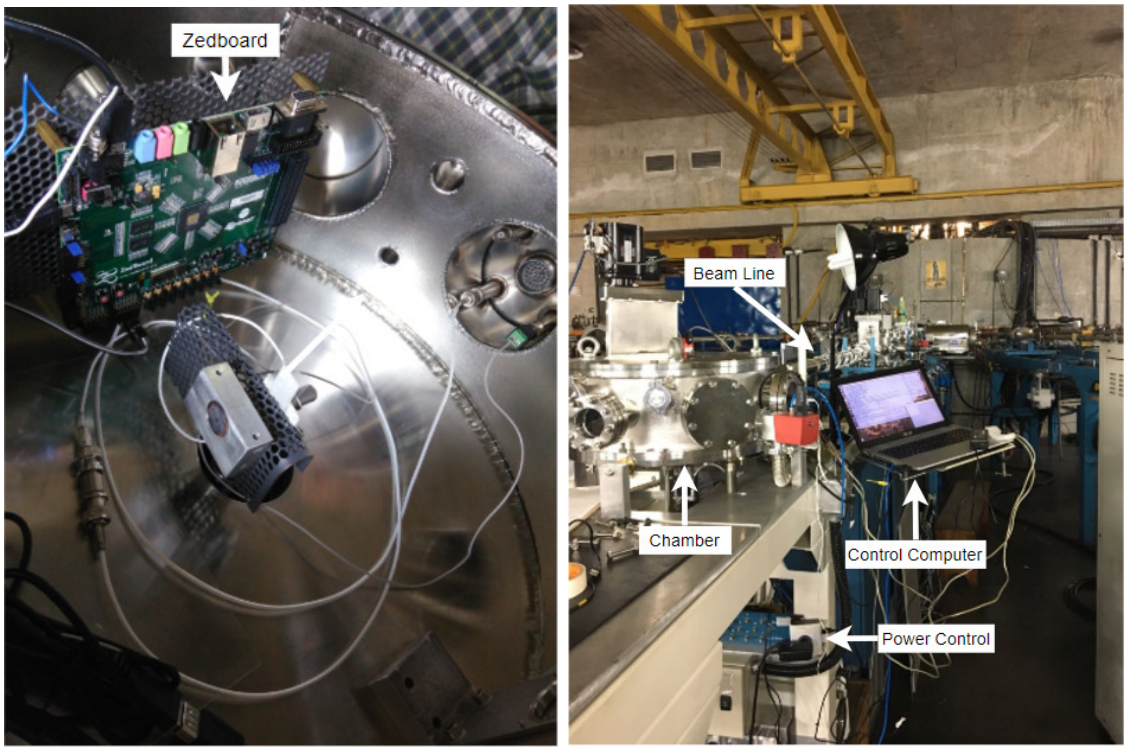


Source: From the author.

Another metric used to compare the system reliability is the Mean Workload Between Failures (MWBF) (RECH et al., 2014; TAMBARA et al., 2016) defined as through Eq. 4.2. The MWBF assess the workload computed correctly until an output error occurs.

$$MWBF = \frac{Workload}{CrossSection \times Flux \times ExecutionTime} \tag{4.2}$$

Figure 4.5: Perspective of radiation experiment setup performed at LAFN-USP: (a) View inside of the chamber; (b) View of the laboratory



(a) (b)

Source: From the author.

4.3 Evaluated Applications

Matrix Multiplication and AES benchmarks were selected to assess the system. Both are extensively used in the literature and well-representative of the general use of the DUT. As software optimizations affects the system performance directly, in this work the applications are also evaluated under distinct optimizations levels.

Different experiments were performed with the applications running in bare-metal, as further detailed in Section 4.3.2. Aiming to evaluate the proposed DCLS efficiency on applications running on top of Operating Systems (OS), the approach also was applied to FreeRTOS systems.

The benchmarks main characteristics are presented below:

- *Matrix Multiplication (MxM)*: In this case, the application is considered as a set of matrix multiplications operations. Each full MxM execution is regarded a block, and between each one there is a verification point. During the experiments distinct array sizes were tested. Each operation multiplies different matrices inputs, which contains data of 32 bits.
- *Advanced Encryption Standard (AES)*: The application is designed to encrypt a vector of 3,200 integers (12,800 bytes). As the maximum size supported by the AES encryption is a vector of 32 integers per time, the total data was divided into 100 blocks of 32 positions. The input vector is set in a loop and the key used is also a vector of 32 integers. Between the encryption of each block there is a verification point, so, there are a total of 101 VPs.

4.3.1 Software Optimizations

To evaluate how the software optimizations impact in performance and error detection and correction of the DCLS approach, the following optimizations are combined: software partition and software compilation.

4.3.1.1 Optimization in Software Partition

To ensure the processor's dependability and to minimize the fault latency, system checks should be performed as often as possible during the application execution. However, the frequency of verifications directly impacts the system performance. The time

required to detect a fault, recover the system to the last checkpoint saved and re-execute the operations severely affects the system efficiency. In this work, the application is partitioned into blocks, aiming to identify a tradeoff that minimizes the performance overhead and error recovering execution time. Each block runs a part of the application, and at the end of each one a VP is set, as detailed in Section 3.3. The number of blocks that the original program is divided can be adjusted depending on the application requirements.

4.3.1.2 Optimization in Software Compilation

The compiler optimizations affect the ARM processor reliability directly, as demonstrated in (LINS et al., 2017). Because the optimizations aim to boost performance, the program code is changed to run efficiently. However, these changes usually concern in use more resources, like registers, which increases the susceptibility to soft errors. Thus, it is imperative to know how the optimizations in software compilation impacts in the DCLS efficiency.

The ARM compiler (ARM, 2014) supports the following optimization levels: O0, O1, O2, and O3. The O0 is the minimal one, where most of the optimizations are disabled. Level O1 has restricted optimizations, producing a good correspondence between source code and object code. The high optimization O2 is the default level in release mode. Level O3 is the highest, and focuses on boosting performance at the expense of code size. The optimizations has a high impact on performance, number of loops and in the program flow execution.

In this work, only the O0 and O3 compiler optimizations were evaluated, considering they represent the levels that use more (O3) and fewer (O0) registers. Because the O3 optimization affects the source code, it is applied only to the block partition. At the verification point, there is a status control that is written just by the processor and read only by the Checker IP. Dead code elimination is one of the code transformations applied by the O3 optimization. Thus, in that case, the status control will be removed from the execution. Compile the whole application with O3 level would interfere with the DCLS approach.

4.3.2 Test Cases Overview

To a complete assessment of the proposed DCLS, distinct test cases are evaluated using Matrix Multiplication and AES benchmarks. Each test case aims to investigate a system particularity and how it impacts in performance and fault mitigation of the DCLS approach. As explained in Section 3.3.1, the Checker IP can verify the processors by comparing all the outputs positions or just the signature. For the MxM test cases, the signature approach is used by making a sum of all the matrix resulting elements before a VP. For the AES, all the output positions are verified.

Table 4.2 presents an overview of the test cases description detailed below:

- **Test case I:** Aiming to investigate the effect of the block size and the number of blocks in the application regarding soft error detection and correction the Matrix Multiplication benchmark is used. Different matrix sizes (3x3, 10x10, and 20x20) are selected to assess the block size. For the number of blocks analysis, two application sizes are considered: short, which is a set of three execution blocks, and long, with six blocks. Thus, the number of verification points is four and seven for short and long application, respectively. All the VPs apply the signature. The benchmark runs in bare-metal, compiled with O0 optimization. This test case is evaluated through fault injection experiments.
- **Test case II:** A set of six blocks of matrix multiplication operations is executed to evaluate the DCLS approach applied to FreeRTOS. Thus, there are seven verification points with signature in the application. The benchmark running on top of FreeRTOS is tested with 40x40 and 60x60 matrix sizes. Also, the bare-metal counterparts are evaluated. For a fair comparison, the application is executed on a single task in the FreeRTOS versions. For all variants, the program is compiled with O0 optimization. The results are gathered by fault injection.
- **Test case III:** In order to analyze the compiler optimizations influence in the designs, an extensive fault injection campaign was performed for AES and Matrix Multiplication applications running in bare-metal and compiled with O0 and O3 optimizations. The MxM is a set of six 40x40 operations, being the signature applied at the VPs. However, for the implemented AES, the signature is not applied at the verification point. The AES application has a total of 101 VPs, as detailed in the begging of this section, and the MxM has seven.

Table 4.2: Test Cases description

Test case	Bench.	Opt.	# Number of blocks	# Block size (bytes)	VP sig.	Version	Experiment
I	MxM	O0	3 and 6	3x3 (36B), 10x10 (400B), 20x20 (1,600B)	Yes	Bare-metal	Fault Injection
II	MxM	O0	6	40x40 (6,400B), 60x60 (14,400B)	Yes	Bare-metal, FreeRTOS	Fault Injection
III	AES	O0 / O3	100	32int. (128B)	No	Bare-metal	Fault Injection
	MxM		6	40x40 (6,400B)	Yes		
IV	MxM	O0 / O3	6	40x40 (6,400B)	Yes	Bare-metal	Radiation

- **Test case IV:** For a realistic evaluation of the DCLS system behavior, radiation experiments were conducted for Matrix Multiplication benchmark. The application is composed of six 40x40 operations running in bare-metal and compiled with both O0 and O3 optimizations. There are seven VPs, and each one applies the signature.

5 RESULTS

This chapter presents the many results of the fault injection and heavy ions experiments performed on Zynq-7000. First, the assessment of the implemented DCLS approach is made, where the results regarding area and performance are analyzed. After, the results gathered by fault injection experiments for four distinct test cases are discussed. Finally, a realistic analysis of the proposed DCLS trough radiation experiments is achieved.

To assess the proposed DCLS in the ARM-A9 processor a series of setups were tested. The test cases, described in Section 4.3.2, are executed considering the designs Unhardened and both DCLS versions. The Unhardened is unprotected against soft errors and runs its application only on ARM CPU0, where the application data is saved on BRAM0. The DCLS designs are the ones detailed in Section 3.3.2: DCLS_BR and DCLS_BR_DDR. These designs use ARM CPU0 and CPU1, two BRAMs and a Checker module implemented in the PL part, besides the DDR. For both Unhardened and DCLS designs, the program instructions are stored in the external DDR memory.

5.1 Implementation Analysis

Aiming to analyze how the DCLS approach impacts in system performance and area resources used, this section gives a complete discussion about the penalties that shall be considered in applying the DCLS to improve the system reliability.

5.1.1 Area assessment

The used resources of each design are detailed on Table 5.1. The DCLS designs use 3,130 Look-Up Tables (LUT) and 3,425 Flip-Flops (FF), which represents an overhead of 275% for LUTs and 244% for FFs, comparing with the unhardened design. The Checker IP requires only 1,298 LUTs (around 41% of the total used number) and 666 FFs (19% of the total). The other resources are required by the AXI interconnect, and BRAM controllers which are necessary to make the communication between the processors and the BRAMs. Given that DCLS uses both processor cores, the overhead concerning the CPUs and memories is 100%. The DCLS_BR and DCLS_BR_DDR designs differ only

at software level, as detailed in Section 3.3.2. Because of that, the hardware is identical, using both the same resources.

The Checker IP implementation does not depend on the processor to be protected. Thus, the resource usage of the Checker IP is fixed. For instance, a typical configuration of the soft-core MicroBlaze processor requires 2,109 LUTs and 1,726 FFs (XILINX, 2017a), which is much more than the resources needed by the Checker IP. Therefore, the impact in use the Checker IP concerning the processor resource usage is low.

Comparing with the related works, the solution in (GOMEZ-CORNEJO et al., 2013) presents an area overhead of 300%. The approach in (PHAM; PILLEMENT; PIESTRAK, 2013) requires 1,104 of extra slices, resulting in 297% of slices usage. The FPGA resources are needed by the following blocks: two MicroBlaze processors; a Voter; three Configuration Engines; a Context Recovery Block: and a Comparator / Multiplexer (COMP_MUX). Besides, it uses 112 KB BRAM memory. The work in (VIOLANTE et al., 2011) uses 8,473 logic cells, which represent 210% of usage. This lockstep architecture uses two soft-core processors, a BRAM memory, and a Checker logic. Because those approaches are implemented using soft-core processors, interconnect and BRAM controllers to communicate the processors and the BRAMs are not required. Thus, there are no extra resources for these blocks as it is demanded in our work. In (ABATE; STERPONE; VIOLANTE, 2008), which is the only related work that uses hard-core processors, is required 5,869 LUTs to implement the lockstep while in our work it is necessary much less. The FPGA blocks needed to implement the approach are a Program Code Controller IP to error detection, two Interrupt IPs, and two interrupt controllers.

Concerning the ARM register file usage, Table 5.2 presents the used registers for both Matrix Multiplication and AES benchmarks, compiled with O0 and O3 optimizations. The MxM with O0 requires a usage of 50% of the registers. On the other hand, this benchmark compiled with O3 uses 25% more registers. For the AES, the version with O3 uses all the registers to execute the application, and the O0 one requires 56% of the register file. As expected, O3 optimization used much more registers, which significantly reduces the execution time, but also increases the susceptibility to soft errors, as it will be further explained. Regarding the DCLS approach, the verification point uses the R0 to R3 and R11 registers. Thus, the only affected version is AES compiled with O0 optimization, which has an increase of 6% of the register file usage. The others application versions are not affected by the proposed DCLS.

Table 5.1: Resource usage of each implemented design

Design	Area (LUTs/FFs)	# CPUs	# BRAMs
Unhardened	833 / 996	1	1
DCLS_BR	3,130 / 3,425	2	2
DCLS_BR_DDR	3,130 / 3,425	2	2

Table 5.2: Description of used registers for each benchmark in different compiler optimizations

Benchmark	Opt.	Used registers	% Reg. file usage
MxM	O0	R0 to R3, R11 / SP, LR, PC	50
	O3	R0 to R7, R12 / SP, LR, PC	75
AES	O0	R0 to R4, R11 / SP, LR, PC	56
	O3	R0 to R12 / SP, LR, PC	100

5.1.2 Performance assessment

The performance results are analyzed comparing the time to perform the applications running in unhardened and DCLS designs. All the obtained results for execution time are expressed in clock cycles (c.c.) for a fault-free execution. In sequence, the specific analysis for each test case is presented.

Concerning the implementation characteristics of the DCLS approach detailed in Section 3.3, the extra execution time in the protected designs is due to several factors, as follows:

- The time required for both processors to execute the application in hand-shake: Due to resource sharing, CPUs need to wait for some resources be available.
- The application size: This aspect impacts the time to perform a checkpoint in the DCLS_BR_DDR designs directly. All the application data saved on the BRAMs is replicated to the DDR. Thus, the time to execute the checkpoint increases proportionally the amount of data stored. Because the BRAMs are the data memories, the application size does not impact in the DCLS_BR designs.
- The output size: In each verification point the outputs of both processors are compared. The amount of data verified directly affects the time to perform the VP.
- The number of verification points performed during the application: The software partition must be defined in a way that the system reliability is assured with low fault latency and the losses in performance are minimized.

5.1.2.1 Test Case I Analysis

Table 5.3 reports the results regarding performance for Matrix Multiplication benchmark with signature running in bare-metal. Different matrix sizes are evaluated: 3x3, 10x10, and 20x20. The 30x30 and 40x40 matrix sizes that are not presented in Test Case I are also evaluated.

Table 5.3: Performance analysis for each design running different matrix sizes

Design	**App. size (# MxM)**	**Matrix sizes**	**Exec. Time (c.c.)**	**Overhead (%)**
Unhardened	Short (3)	3x3	98,280	-
		10x10	2,207,018	-
		20x20	15,763,012	-
		30x30	51,343,884	-
		40x40	119,758,930	-
	Long (6)	3x3	182,876	-
		10x10	4,291,642	-
		20x20	30,827,142	-
		30x30	99,950,676	-
		40x40	231,917,552	-
DCLS_BR	Short (3)	3x3	516,296	425.3
		10x10	2,930,374	32.8
		20x20	19,896,474	26.2
		30x30	63,689,580	24.0
		40x40	148,497,868	23.6
	Long (6)	3x3	707,732	287.0
		10x10	5,780,740	34.7
		20x20	38,618,858	25.3
		30x30	124,760,012	24.8
		40x40	287,243,296	23.9
DCLS_BR_DDR	Short (3)	3x3	712,544	625.0
		10x10	4,088,692	85.3
		20x20	23,319,986	47.9
		30x30	71,256,642	38.8
		40x40	161,541,952	34.7
	Long (6)	3x3	1,009,842	452.2
		10x10	7,336,424	70.9
		20x20	44,173,342	43.3
		30x30	137,032,766	37.1
		40x40	311,447,120	34.3

As described in Table 5.3, the performance overhead is significantly higher, around 425% to DCLS_BR and 625% to DCLS_BR_DDR (short version of 3x3 matrix), when the execution time of the benchmark is much smaller compared to the time to perform a verification point. For large applications, the time overhead of DCLS_BR is less than 25%, which can be an acceptable cost for many applications that require high reliability and availability. When considering the setup DCLS_BR_DDR the time overhead in all versions is higher compared to DCLS_BR, as expected, owing to the time to access the DDR memory.

The execution time overhead reported in (VIOLANTE et al., 2011) ranges from 17% to 54% depending on the amount of data saved on a checkpoint. The achieved DCLS results show data close to this for the largest matrices. For sizes bigger the ones tested, the performance overheads are expected to be lower.

5.1.2.2 Test Case II Analysis

The performance results for the Test Case II with the applications running on top of FreeRTOS and in bare-metal are presented in Table 5.4. In the Test Case II, the benchmark is a set of six Matrix Multiplication operations with 40x40 and 60x60 sizes evaluated. Comparing the FreeRTOS versions with the bare-metal counterparts, one can notice that the former has few impact in the performance. When using FreeRTOS, the system under evaluation becomes more complex. The extra time is due to the OS task management, in which more instructions are executed. For both the bare-metal and FreeRTOS series of setups, the results show a variation of around 7% regarding performance overhead between the DCLS_BR and DCLS_BR_DDR. This is due to the saving of data in the DDR memory.

Table 5.4: Performance analysis for each design running bare-metal and on FreeRTOS

Version	**Design**	**Matrix 40x40**		**Matrix 60x60**	
		Exec. Time (c.c.)	**Overhead (%)**	**Exec. Time (c.c.)**	**Overhead (%)**
Bare-metal	Unhardened	233,013,420	-	773,865,026	-
	DCLS_BR	299,233,218	28.41	990,731,336	28.02
	DCLS_BR_DDR	312,527,578	34.11	1,048,091,310	35.43
FreeRTOS	Unhardened	234,217,175	-	774,615,876	-
	DCLS_BR	303,571,104	29.61	996,395,350	28.63
	DCLS_BR_DDR	325,504,854	35.98	1,050,843,464	35.65

5.1.2.3 Test Case III Analysis

Table 5.5 presents the performance results of each design running the Matrix Multiplication (40x40) and AES benchmarks with O0 and O3 compiler optimizations in bare-metal. For the ones that execute the applications compiled with the O0 optimization, the overhead is close to the results achieved for Test Cases I and II previously presented. Because the benchmark with O3 runs faster than with O0, and the verification point is not compiler optimized, the overhead for these applications is higher.

Concerning the DCLS_BR_DDR, the cost for O3 optimization cases is significant, as the time to save all data in the DDR is far greater than execute the optimized block partition. As one can notice, for O3 MxM DCLS_BR_DDR the overhead is enormous (548%), while the O3 AES for the same design is about 78%. Because of the applications structure, as detailed in Section 4.3, the MxM has a significant amount of data that must be saved regarding the input matrices and results.

5.1.2.4 Software Partition Evaluation

Owing to the high performance overhead achieved in the test case results, an impact evaluation of using the proposed DCLS is required. The main drawback of the DCLS approach presented in this work is related to the processors verification and checkpoints (context saving). For several applications, like the real-time ones, performance penalties are critical. Nevertheless, for other applications, a low overhead is acceptable given the system reliability improvement. The aiming is achieving the best tradeoff between an acceptable overhead and system protection.

Table 5.5: Performance analysis for each design performing Matrix Multiplication (40x40) and AES benchmarks compiled using O0 and O3 optimizations

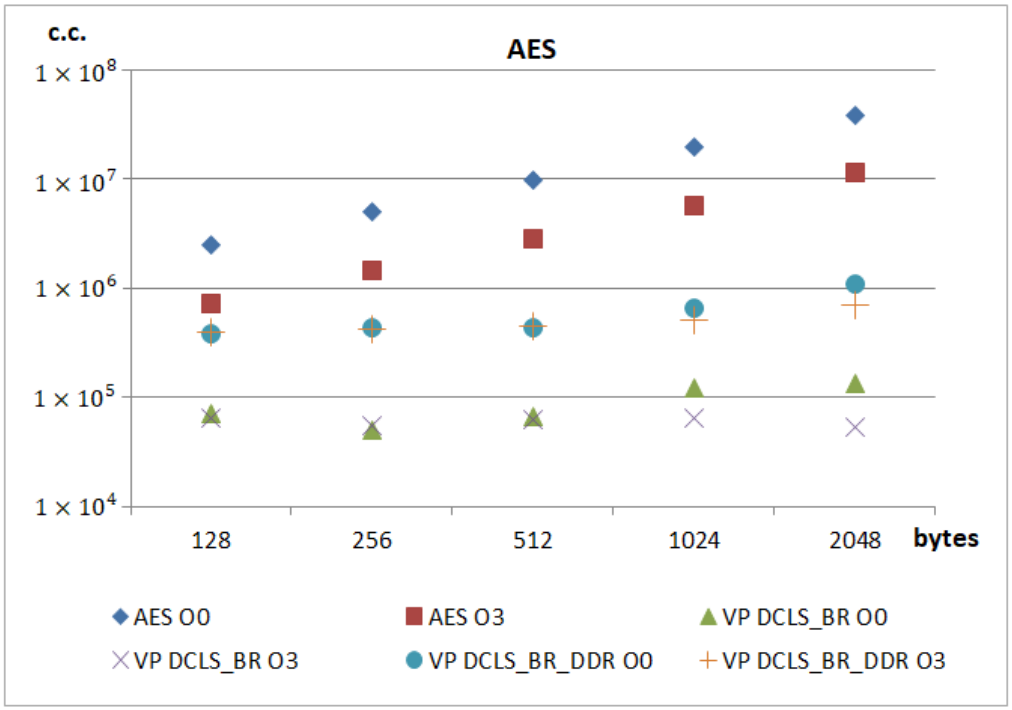
Design	**Opt.**	**MxM**		**AES**	
		Exec. Time (c.c.)	**Overhead (%)**	**Exec. Time (c.c.)**	**Overhead (%)**
Unhardened	O0	233,013,420	-	206,834,492	-
	O3	4,881,780	-	62,361,916	-
DCLS_BR	O0	299,233,218	28.41	261,216,708	26.29
	O3	7,191,982	47.32	79,636,072	27.70
DCLS_BR_DDR	O0	312,527,578	34.12	290,548,962	40.47
	O3	31,647,168	548.27	110,903,870	77.84

Figures 5.1, 5.2 and 5.3 present the relation between the execution time (in clock cycles) to perform the benchmarks and the verification point, for a given amount of outputs data in bytes. The considered benchmarks are MxM and AES compiled in both O0 and O3 optimizations. The graphics show the execution time for performing the block partition and the VP for DCLS_BR and DCLS_BR_DDR designs.

Fig. 5.1 presents the execution of AES with different block partitions: 1, 2, 4, 8, and 16 sequential encryptions of 32 integers, which represents 128, 256, 512, 1024, and 2048 total bytes, respectively. From this figure, one can notice that for both DCLS designs the time to perform the AES for all the block sizes tested in both optimizations is greater than the execution time of the VPs. However, it does not guarantee a small performance overhead. Regarding the block partitions with small size, it is notable that the DCLS_BR_DDR has an execution time close to the benchmark, which leads to a high overhead, as shown in Table 5.5. This extra time corresponds to the data saving in the DDR and, also, the access to the DDR memory is slower than the BRAM. On the other hand, the execution time of the VPs in DCLS_BR designs is much smaller than the benchmark, which leads to an acceptable overhead. This graphic shows that increasing the block partition size the VP execution overhead decreases.

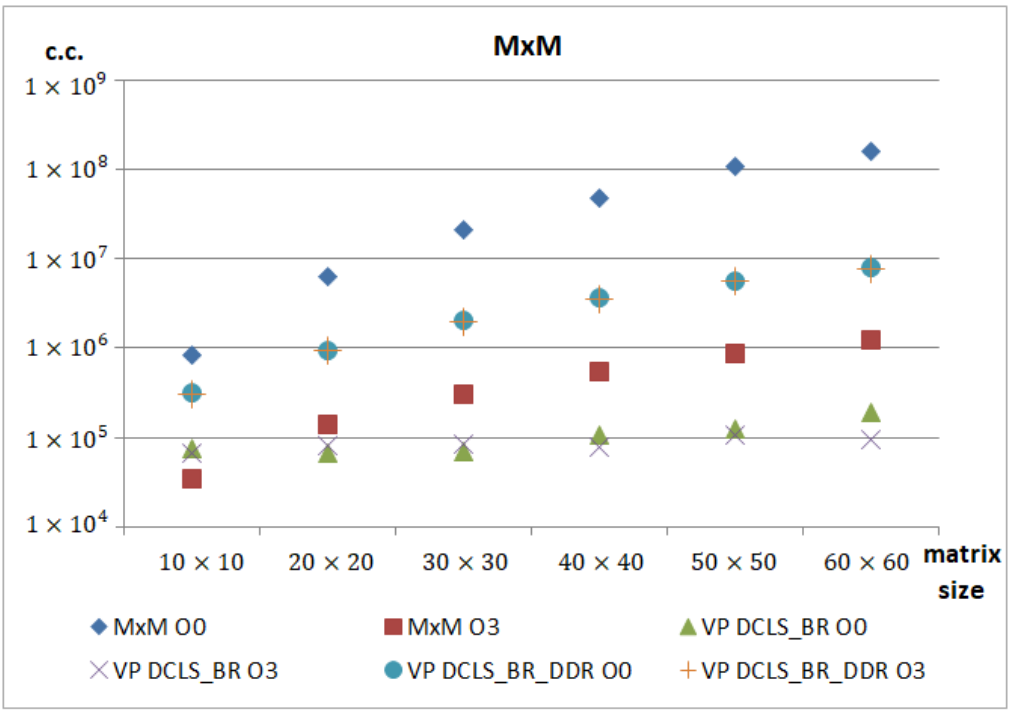
The results concerning matrix multiplication are presented in Fig. 5.2. The assessment is made for 10x10, 20x20, 30x30, 40x40, 50x50, and 60x60 matrix sizes. Analyzing data size, the VP execution for the small matrices for both DCLS designs has a high impact on performance. As the matrix size increases, for the O0 optimization, the VP time in both DCLS cases affects less the performance, like in the AES. Regarding the O3 optimization, the DCLS_BR has the same behavior, but with more performance impact. Because the O3 application is faster than the O0, and the optimization is not applied to the VP, the execution time difference is still small. However, the tendency for this difference is to increase for bigger matrices, achieving better performance results. On the other hand, this tendency does not appear in the DCLS_BR_DDR for the O3 optimization. In this case, the execution time of the VP is always longer than the execution of the MxM. Saving the VP data on the DDR severely affects the performance.

Figure 5.1: Execution time in clock cycles (c.c.) for performing the AES block partitions and the Verification Points



Source: From the author.

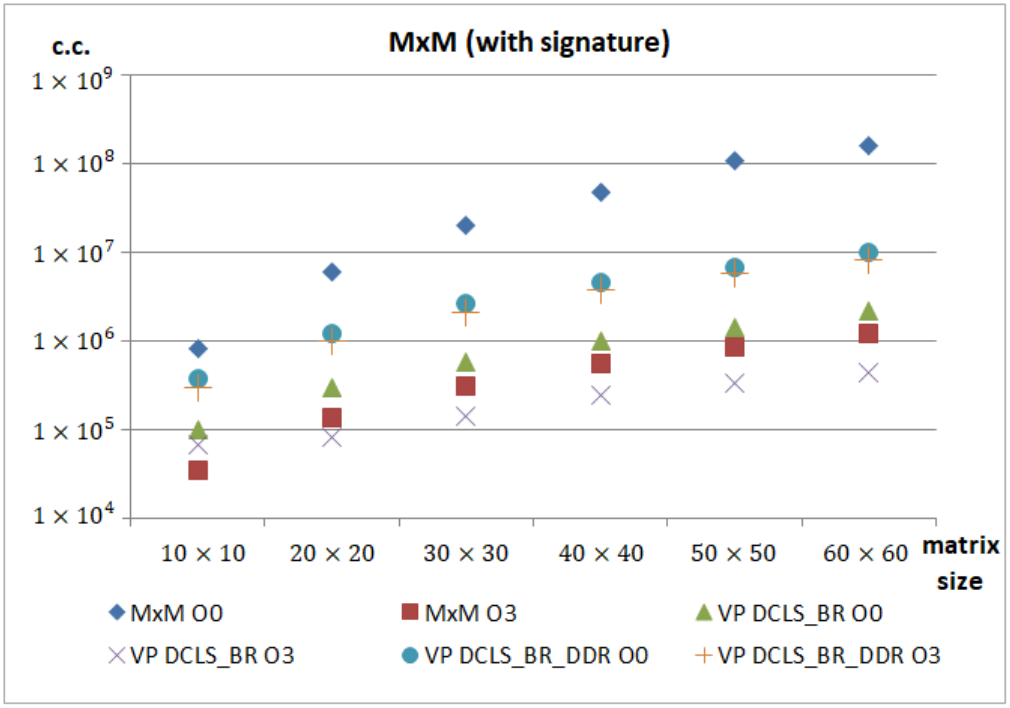
Figure 5.2: Execution time in clock cycles (c.c.) for performing the Matrix Multiplication in different sizes and the Verification Points



Source: From the author.

Another approach for the verification point implementation is based on signatures, as presented in Section 3.3. Fig. 5.3 shows the execution time for matrix multiplication and DCLS designs with signature. In the verification point, a signature is previously applied to the outputs. Thus, the Checker compares just one memory position corresponding to the signature. Before the processor be locked on the VP, a sum of all elements of the resulted matrix is performed, and instead of all output positions be compared by the Checker, only the signature is compared. The sum of the elements is also performed in both O0 and O3 optimizations, which directly affects the execution time of VP. For the DCLS_BR_DDR, the results compared with Fig. 5.2 without the signature are almost the same, as the time to save the amount of data on DDR is much larger than calculating the signature. Analyzing the results for matrices 40x40 with O3 presented in Table 5.5, it is noticed that using DCLS_BR_DDR produces a huge overhead in the final application. Concerning the DCLS_BR designs, the overhead is acceptable, however applying the signature affects more the performance.

Figure 5.3: Execution time in clock cycles (c.c.) for performing the Matrix Multiplication in different sizes and the Verification Points with signature



Source: From the author.

5.2 Fault Injection Experiments

To assess the impact of soft errors in a dual-core ARM processor and to validate the efficiency of the proposed DCLS approach, an extensive fault injection campaign was performed in the Zedboard. The fault injection experimental methodology is presented in Section 4.1. All the four presented test cases are evaluated in Unhardened and DCLS designs.

5.2.1 Evaluation of Test Case I

In the Test Case I, Matrix Multiplication with signature in bare-metal is evaluated for different block partitions. Table 5.6 shows the fault injection results for each design running 3x3, 10x10 and 20x20 matrix sizes. For these experiments, there is no log distinction between the masked faults, mitigated faults and UNACE. Thus, the UNACE column represents that the injected bit-flip did not affect the system or the fault was detected and corrected by the DCLS approach.

For the Unhardened versions, up to 70% of the injected faults are classified as UNACE. For the DCLS designs, this number increases to around 91% in the DCLS_BR and 90.5% in DCLS_BR_DDR design. The injected faults that produce SDCs (wrong outputs values) are up to 13% for Unhardened, while they are negligible for the DCLS designs. Even so, the SDCs in the DCLS can be explained by bit-flips in the LR or PC registers that can direct the program pointer to the end of the application. Thus, when the outputs results are compared with the gold ones, they mismatch, and an SDC is indicated. Even in the worst case, when is achieved 1.3% of SDCs in the DCLS_BR, the approach is able to reduce almost 11% the errors. Therefore, the effectiveness of the proposed DCLS in detecting and correcting errors is confirmed.

Up from 8% of the bit-flips that could not be recovered provoke hangs in the DCLS system. This result can be explained by two facts. First, there are some registers that could not be protected by the proposed DCLS, as detailed in the Section 3.3.1. Therefore, during an execution block, a fault can upset one of those registers. For some reason, the bit-flip effect can be masked, and it does not affect the outputs. However, the register still has a wrong value. In the verification point, the Checker will not be able to detect this fault, which leads to storing the actual wrong context as a safe state. Thus, the fault can manifest itself in the next execution block leading a rollback operation that will restore the

Table 5.6: Fault injection analysis for each design running different matrix sizes for Test Case I

Setup			Fault Injection		
Version	**App. size (# MxM)**	**Matrix sizes**	**UNACE (%)**	**SDC (%)**	**Hang (%)**
Unhardened	Short (3)	3x3	69.6	12.0	18.4
		10x10	64.8	8.5	26.7
		20x20	67.0	13.2	19.8
	Long (6)	3x3	66.3	8.3	25.4
		10x10	63.6	7.3	29.1
		20x20	64.6	9.1	26.3
DCLS_BR	Short (3)	3x3	90.9	1.3	7.8
		10x10	88.3	0.0	11.7
		20x20	88.3	0.0	11.7
	Long (6)	3x3	90.8	0.2	9.0
		10x10	88.2	0.0	11.8
		20x20	89.4	0.0	10.6
DCLS_BR_DDR	Short (3)	3x3	86.4	0.1	13.5
		10x10	86.4	0.2	13.4
		20x20	90.2	0.4	9.4
	Long (6)	3x3	90.5	0.1	9.4
		10x10	88.6	0.0	11.4
		20x20	90.1	0.0	9.9

wrong context causing, then, an infinite loop in the system. Second, if a fault affects any of the specific registers (SP, LR or PC), generating an illegal data or instruction value, the processor will be directed to data or prefect abort leading to a system crash. The hang or timeout can be identified, but only can be recovered by reset and, for these experiments, the soft reset had not been implemented yet.

5.2.2 Evaluation of Test Case II

Aiming to evaluate the DCLS technique applied to FreeRTOS the Test Case II is used. There are two versions of each design: bare-metal and FreeRTOS. The 40x40 and 60x60 sizes are evaluated for Matrix Multiplication benchmark with signature in the VP. As detailed in Section 4.1, for the bare-metal cases a fault can be injected at any time during the benchmark execution. In the FreeRTOS, a bit-flip can affect the system since

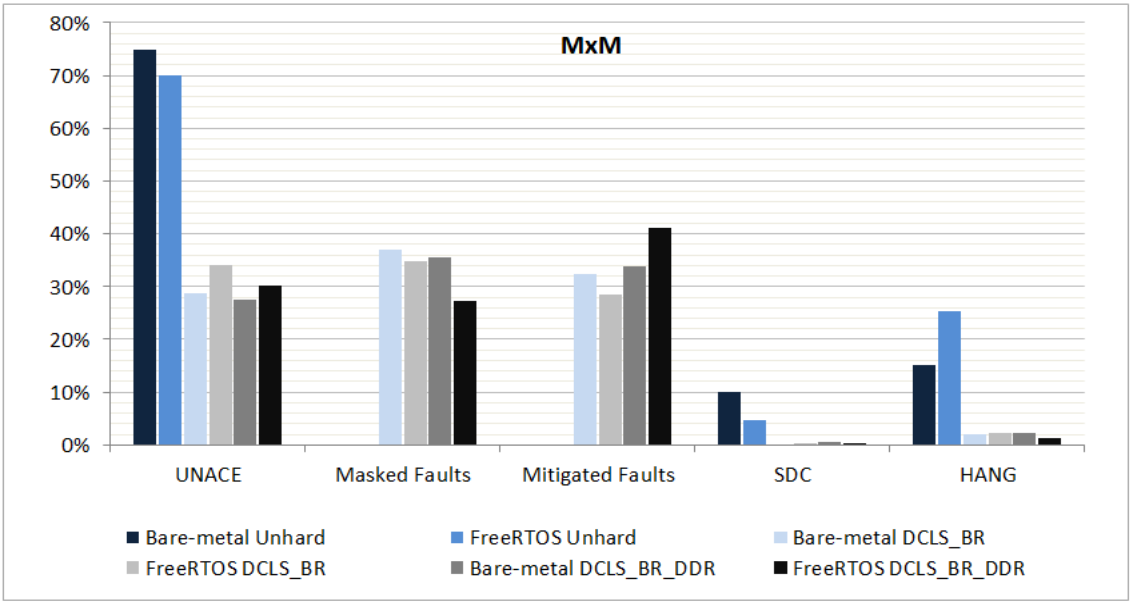
the task configuration until the end of the task execution. The percentage of errors for each experimental design is shown on Figures 5.4 and 5.6, for 40x40 and 60x60 matrix sizes, respectively.

For the Unhardened versions, up to 75% of bit-flips does not affect the system and are classified as UNACE. However, the unprotected designs are very susceptible to SDCs and hangs. Comparing the unprotected FreeRTOS designs with the bare-metal counterparts, one can observe that in the former the faults leads to more hangs than in the latter. Due to the task management and OS control, the FreeRTOS is more complex than bare-metal, and if a fault affects a register used for controlling the system, this may lead to crash. These results also are achieved in (RODRIGUES; KASTENSMIDT, 2017) that evaluated FreeRTOS applications under fault injection simulation and concluded that benchmarks running on FreeRTOS are much more susceptible to hangs than their bare-metal counterparts.

The results show that the DCLS applied to FreeRTOS system can protect, which represents the sum of Masked and Mitigated Faults, against up to 68% of the injected faults. For the bare-metal ones, this number increases to up to 71%. Besides, the quantity of SDCs and hangs for DCLS designs in both versions are almost the same. Thus, using FreeRTOS has a low impact on the DCLS functionality.

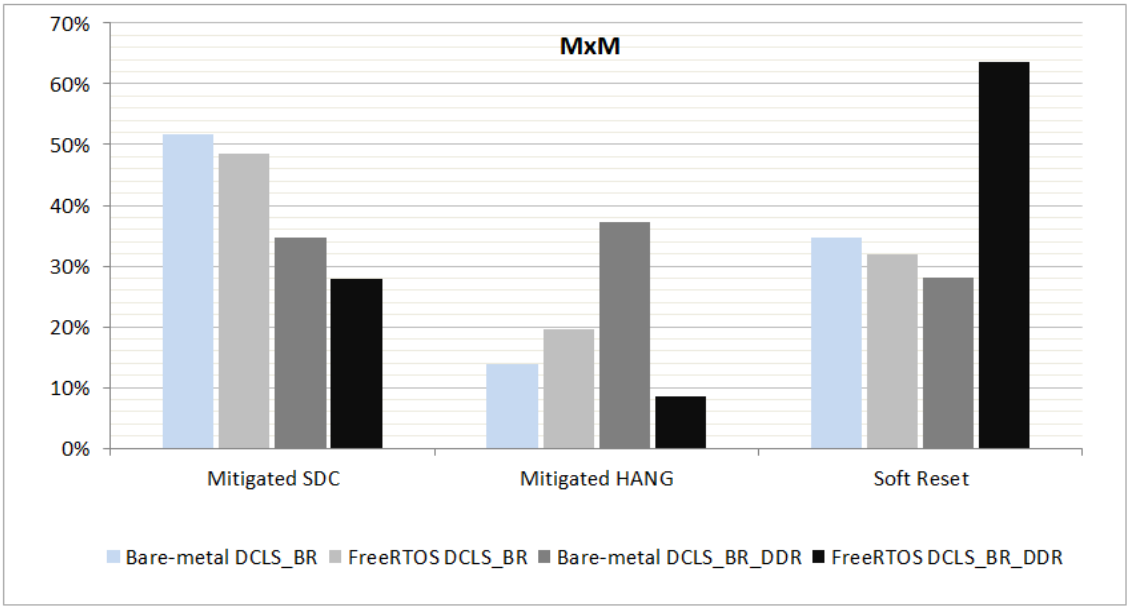
Figures 5.5 and 5.7 detail the distribution of the mitigated faults for 40x40 and 60x60 matrix sizes, respectively. The Mitigated Faults are split in: Mitigated SDC, which are the outputs errors that are successfully corrected by rollback; Mitigated Hang, represents the crashes that are also corrected by rollback; and the Soft Reset, which are the hangs that can only be corrected through a reset in the system. The results show that for FreeRTOS versions the mitigated SDCs ranges from 28% to 55%. While for bare-metal its ranges from 18% to 52%. For the FreeRTOS designs occur more system crashes that lead to soft reset than the ones corrected by rollback. If a fault affects a control register, it may crash the FreeRTOS and the only possible recovery is by reset.

Figure 5.4: Percentage of errors classification for Test Case II experimental designs with 40x40 Matrix Multiplication benchmark



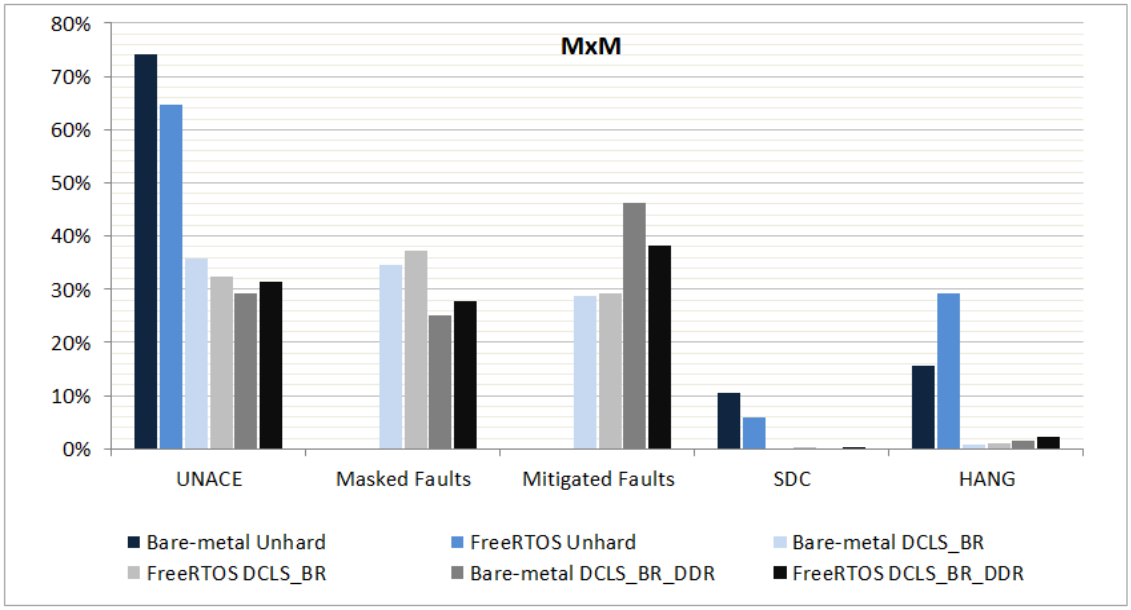
Source: From the author.

Figure 5.5: Distribution of mitigated faults in DCLS designs for Test Case II with 40x40 Matrix Multiplication benchmark



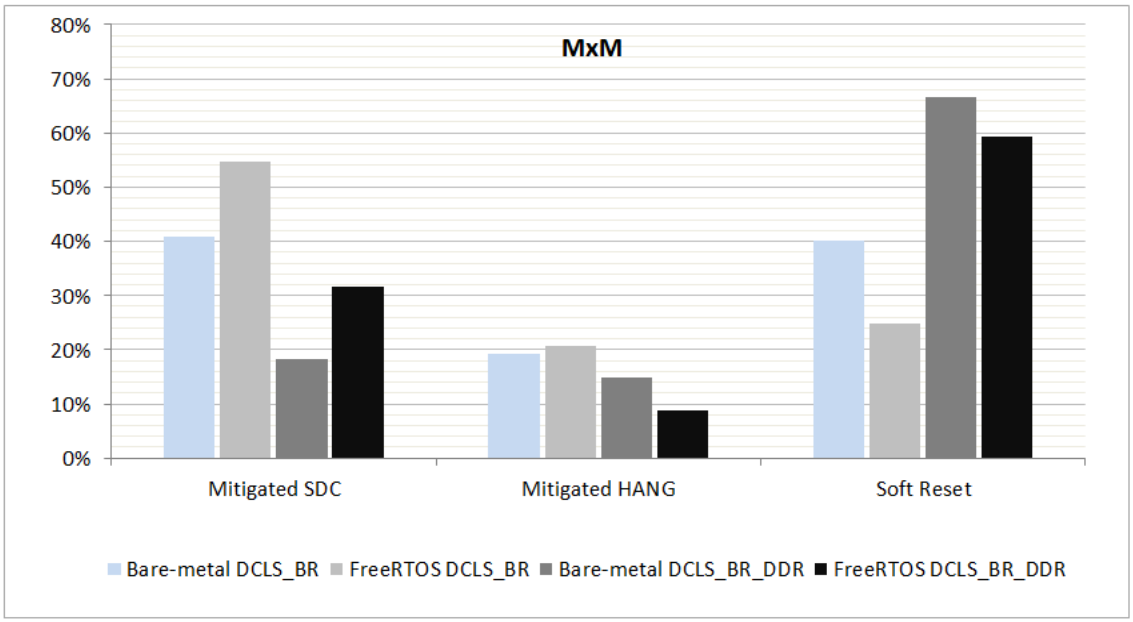
Source: From the author.

Figure 5.6: Percentage of errors classification for Test Case II experimental designs with 60x60 Matrix Multiplication benchmark



Source: From the author.

Figure 5.7: Distribution of mitigated faults in DCLS designs for Test Case II with 60x60 Matrix Multiplication benchmark



Source: From the author.

5.2.3 Evaluation of Test Cases III

Aiming to investigate the compiler optimization influence in the DCLS protection, both AES and Matrix Multiplication (40x40) applications are evaluated. The benchmarks run in bare-metal and are compiled with O0 and O3 optimizations. The fault injection results are presented in Figures 5.8 and 5.10. Analyzing the results, one can notice that the DCLS protects (sum of Masked and Mitigated Faults) against up to 78% and 62% of the injected faults for MxM and AES, respectively. Even in the worst cases (69% for MxM and 50% for AES), the DCLS provides high protection.

To better analyze the mitigated faults by the DCLS approach, Figures 5.9 and 5.11 present the distribution of the corrected errors for AES and MxM, respectively. As previously mentioned, the Mitigated Faults are divided in: Mitigated SDC; Mitigated Hang; and the Soft Reset. The results show that for AES in all versions the mitigated SDCs are around 45%, while for MxM this number ranges from 11% to 51%. The number of corrected hangs for both benchmarks demonstrates that DCLS_BR_DDR is more susceptible to hangs than the DCLS_BR version. Although using the DDR increases the data reliability, this enhancement is not perceptible in the fault injection experiments. As the faults only affect the processor's register file, the data stored in the BRAMs are not susceptible to bit-flips. Because using the DDR increases the application execution time, these designs are more vulnerable to errors than the DCLS_BR. However, in a real radiation environment, the DCLS_BR_DDR designs are expected to be more reliable, because the whole device would be exposed to errors.

When the rollback does not correct the system crash, a soft reset is performed. If a bit-flip affects any of the specific registers, generating data or instruction illegal value, it can cause a system crash that leads the ARM processor to stop handling interruptions. Thus, the Checker identifies a hang and indicates the system to rollback, but only one CPU can process that. Therefore, after successive fails rollbacks, the system is forced to restart by a soft reset.

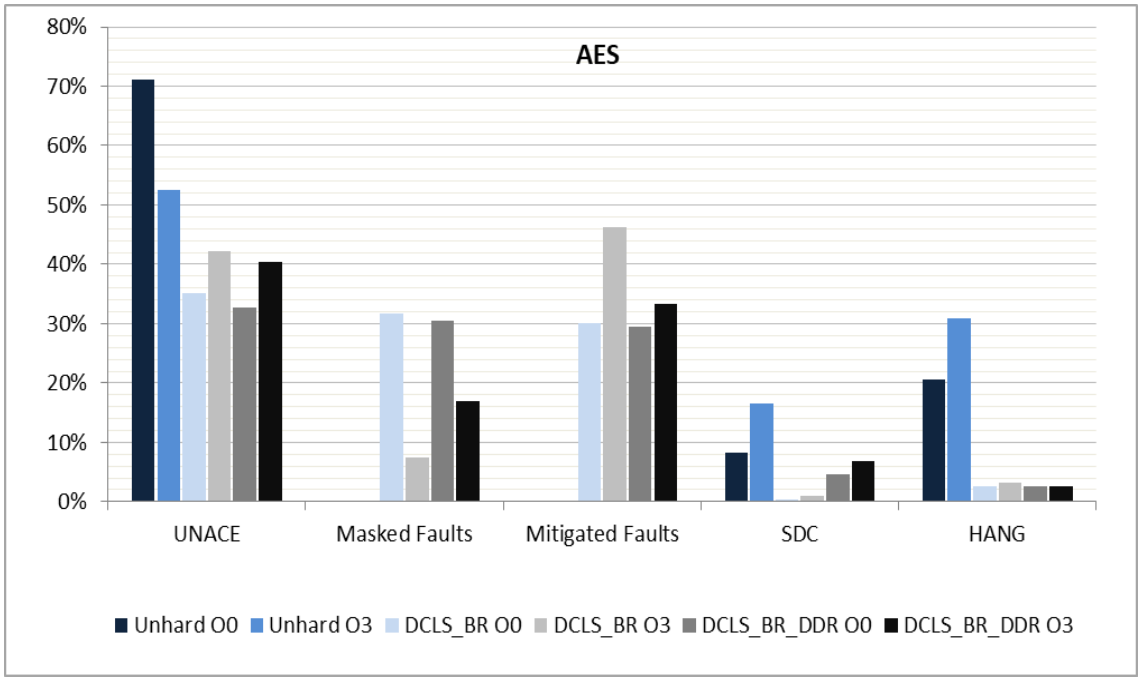
From the results in Fig. 5.10, one can observe the faults that lead to SDCs in the final application are negligible in the MxM cases, as for the Test Cases I and II experiments previously presented. However, for the AES DCLS_BR_DDR designs, they are considerable, as shown Fig. 5.8. These SDCs may happen because of errors in the loop control when the rollback is performed from the DDR: at the end of the application, even if the data is correct, it can be saved in a wrong memory position, causing a mismatch during

the checking phase. Moreover, the DCLS approach is not able to correct all crashes. The injected fault can lead to persistent system hangs. A bit-flip can affect a critical register leading the processor to a crash that is unrecoverable, even with a soft reset. For these permanent faults, a hard reset is necessary.

The use of O3 optimization in AES benchmark affected the protection rate of the DCLS designs. The most affected case is the AES DCLS_BR_DDR, having a protection drop from 60% on O0 to 50% on O3. Analyzing the compiler optimization effects in the Unhardened designs, the ones with O3 level appear as more susceptible to SDCs and hangs, as expected. Because the O3 handles more registers than O0, as shown Table 5.2, the applications with O3 optimization are more vulnerable to soft errors. Therefore, the probability of injected faults leading to errors increases with optimizations.

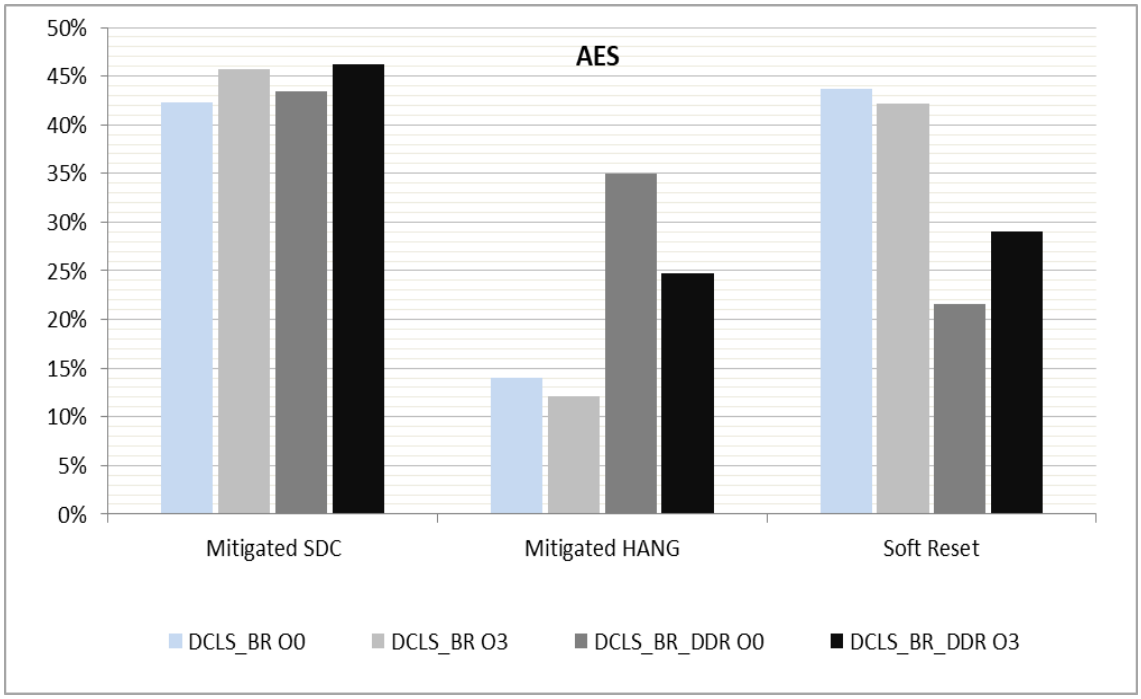
Both (VIOLANTE et al., 2011) and (ABATE; STERPONE; VIOLANTE, 2008) injected faults at processor's registers. The results presented in (VIOLANTE et al., 2011) show that their technique can detect and correct 20% of the injected bit-flips, 1% leads to hang, and 79% of the faults are effectless. Because the faults are only injected in the pipeline registers of the soft-core processor, it is straightforward the fault detection and recovery implementations. In (ABATE; STERPONE; VIOLANTE, 2008), the authors concluded that 97% of the faults do not cause application errors (where up to 54% are corrected, and the others do not affect the system) and 3% provoked SDCs. In experiments, they did not have hangs, which is a curious result as the faults are also injected in the specific and control-flow registers. On this scenario, the authors did not implement a time-out watchdog monitor in the system. Differently, our work is capable of both detecting and correcting hangs.

Figure 5.8: Percentage of errors classification for Test Case III experimental designs with AES benchmark compiled with both optimizations



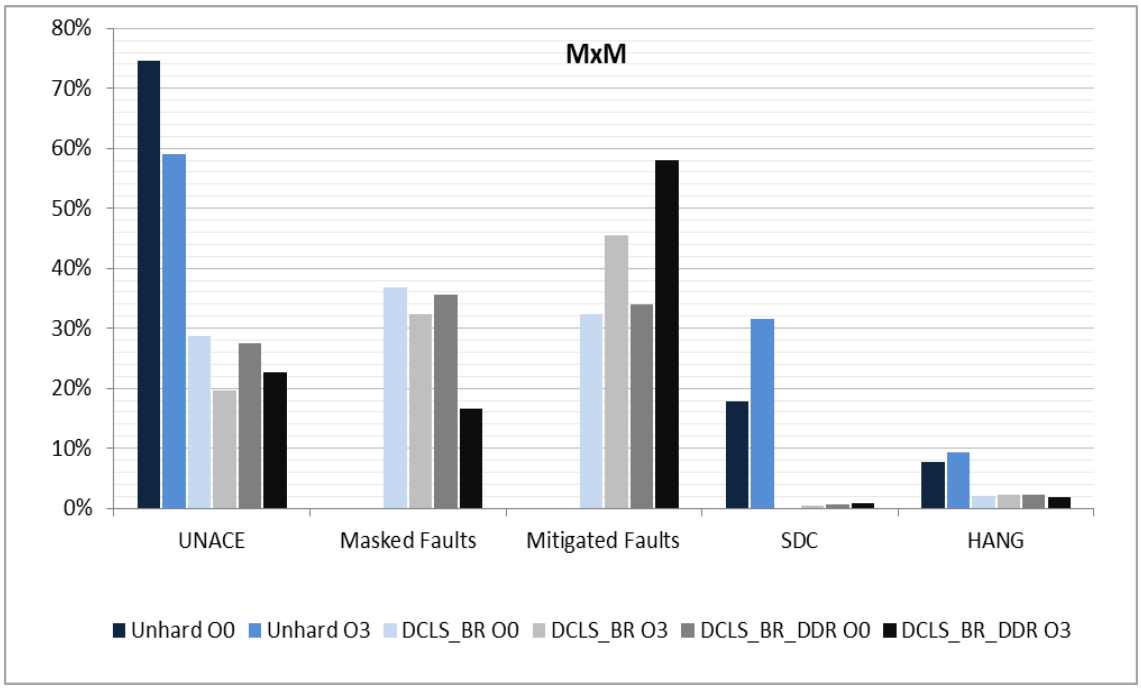
Source: From the author.

Figure 5.9: Distribution of mitigated faults in DCLS designs for Test Case III with AES benchmark compiled with both optimizations



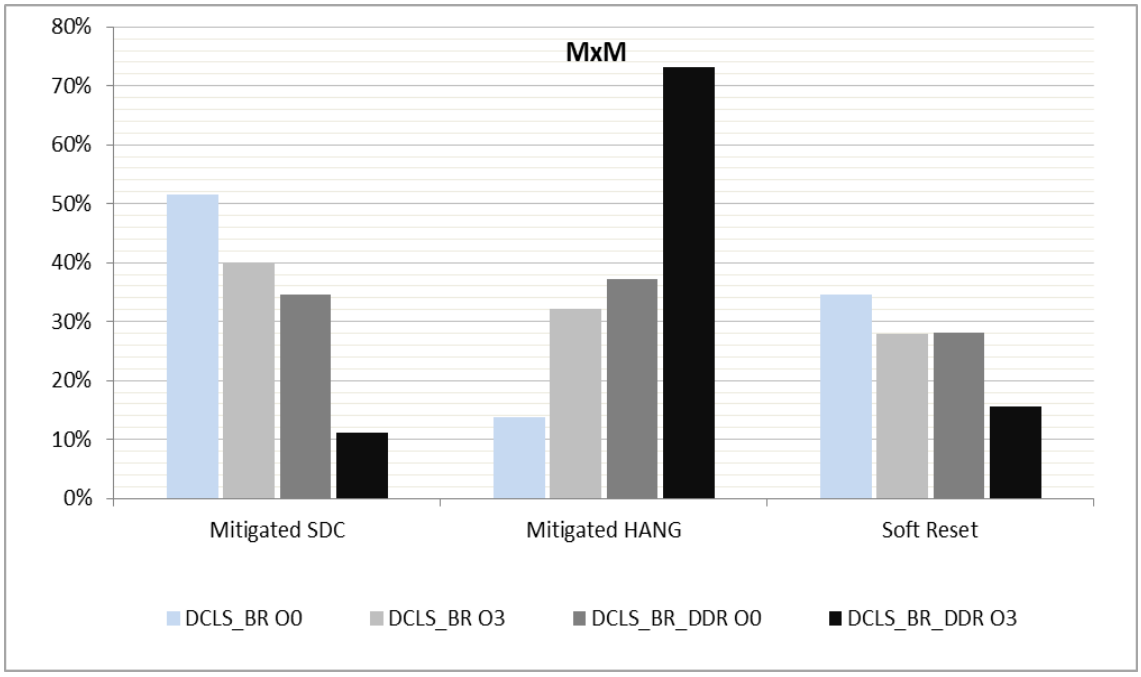
Source: From the author.

Figure 5.10: Percentage of errors classification for Test Case III experimental designs with 40x40 Matrix Multiplication benchmark compiled with both optimizations



Source: From the author.

Figure 5.11: Distribution of mitigated faults in DCLS designs for Test Case III with 40x40 Matrix Multiplication benchmark compiled with both optimizations



Source: From the author.

5.3 Radiation Experiments

For the radiation experiments, the Unhardened and DCLS_BR_DDR designs were tested with Matrix Multiplication 40x40 compiled in O0 and O3 optimizations in bare-metal (Test Case IV). The experimental methodology for the heavy ions tests is presented in Section 4.2. As the limited time to execute the experiments at LAFN-USP Pelletron accelerator, just the Test Case IV was evaluated under radiation.

5.3.1 Evaluation of Test Case IV

The results of the radiation experiments are shown in Table 5.7. Comparing the DCLS with the Unhardened designs, one can notice that the DCLS technique reduces the total cross section, which demonstrates the effectiveness of the approach. Due to the implemented methods to decrease the accumulation of bit-flips in the device, the number of SEFIs are negligible.

Analyzing the Mean Workload Between Failures (MWBF), which describes the amount of data computed correctly before a failure occur, the DCLS approach achieves an improvement of one order of magnitude for O0 optimization. A higher MWBF means a more reliable system. Concerning the O3 optimization, the MWBF is almost the same for both designs. Because the MWBF assesses the tradeoff between error rate and performance, and the execution time to perform the DCLS_BR_DDR design with O3 is around six times the Unhardened one (Table 5.5), the reliability improvement is masked in this case. As demonstrated in Section 5.1.2, to increase the system's reliability with minimal performance losses it is necessary that the execution time of the application block be much longer than the time to perform a verification point. Thus, other block partitions must be analyzed for the O3 DCLS_BR_DDR design in order to increase the MWBF.

Although using O3 optimization in software compilation increases the register susceptibility, this optimization leads to higher MWBF because the execution time is drastically reduced. Comparing only the optimization effects in DCLS and Unhardened designs, the ones with O3 increase the MWBF in one and two orders of magnitude, respectively.

Table 5.7: Experimental results from the heavy ions test campaign in Zynq-7000 device for Test Case IV with Matrix Multiplication (40x40)

Design	**Opt.**	**Fluence** (p/cm^2)	**Flux** ($p/cm^2/s$)	**Cross Section** (cm^2)	**MWBF** (data)
Unhardened	O0	5.04×10^5	2.84×10^2	7.15×10^{-5}	4.33×10^7
	O3	3.83×10^5	2.19×10^2	6.79×10^{-5}	2.82×10^9
DCLS_BR_DDR	O0	1.44×10^6	4.07×10^2	5.54×10^{-6}	2.90×10^8
	O3	1.42×10^6	3.93×10^2	9.18×10^{-6}	1.79×10^9

6 CONCLUSION

This work explored the use of Dual-Core LockStep (DCLS) as a fault tolerance solution to increase the dependability in hard-core processors embedded into APSoCs. Lockstep is a hybrid technique based on redundancy capable of mitigating radiation-induced soft errors. The method is able to detect and recover both SDC and SEFI errors through the combination of outputs verification, and checkpoint and rollback operations. As case study, the proposed DCLS was designed and implemented to protect a dual-core ARM Cortex-A9 processor embedded into Zynq-7000 APSoC from Xilinx. Although the implementation focuses on the ARM processor, the approach can be extended to different hard processors embedded into APSoCs with a few adjustments.

The approach efficiency was validated through distinct test cases using Matrix Multiplication and AES benchmarks. This work applies the DCLS to applications running not only in bare-metal but also on top of FreeRTOS. To assess the impact of using the DCLS to the system two types of software optimizations were evaluated: software partition and compiler optimization. In the former, different application block sizes were evaluated in order to quantify the contribution of the verification points to the overall application performance. For the latter, the O0 and O3 compiler optimizations, which are the most representative levels, were performed to understand the different implications of compiler optimization to the performance and error mitigation of the DCLS approach.

Additionally, to evaluate the fault detection and mitigation of the proposed DCLS, fault injection emulation was performed. Bit-flips were randomly injected in the processors' registers to simulate soft errors. Results show the effectiveness of the proposed DCLS in mitigating up to 78% of the injected faults in the bare-metal test cases. For the FreeRTOS versions, the DCLS successfully corrected up to 68% of bit-flips, which implies that using FreeRTOS has a low impact on the DCLS functionality. Thus, the DCLS can also be applied to applications running on top of FreeRTOS without losing the effectiveness. Comparing with bare-metal counterparts, the FreeRTOS variants are more susceptible to hangs than SDCs due to the operating system management.

Heavy ions experiments were performed for a realistic evaluation of the system. The tests were conducted in the 8UD Pelletron accelerator at LAFN-USP. The obtained results show that the DCLS approach is able to decrease the system cross section with a high rate of protection. Moreover, the technique was able to increase the MWBF, which demonstrates that the DCLS system is more reliable than the unprotected one.

The performance analysis demonstrated that the execution time of the application block must be much longer than the period to perform a verification point, increasing the system reliability with fewer performance penalties. If the time to verify the processors and perform a checkpoint is close to the block execution time, the impact in performance is significantly high. However, for longer block period the cost can be accepted due to increasing the system resilience to soft errors. Although the high overhead, the verification point strategy brings the possibility of each processor works with a distinct clock. Besides, the performance analysis revealed that applying outputs signature in the processor impacts more than comparing all memory positions by the Checker IP.

The assessment of the compiler optimizations demonstrated that the application compiled with O3 level is more susceptible to soft errors. The system is more vulnerable to faults because the O3 optimization uses more registers than other levels. Although the system vulnerability increases, the results from heavy ions experiments show a higher MWBF for the O3 level. Because the application runs faster, more data are correctly computed before an error occurrence.

The main conclusions are summarized below:

- **Block partition size:** The block execution time must be longer than the verification point.
- **Signature:** Applying outputs signature impacts more the performance.
- **Compiler Optimization:** The application compiled with O3 level is more susceptible to soft errors. However, it leads to higher MWBF.
- **FreeRTOS:** The FreeRTOS system is more susceptible to hangs than bare-metal.
- **Protection:** The DCLS approach is able to mitigate a high number of faults in both bare-metal and FreeRTOS systems.

6.1 Future Work

The presented work demonstrated the radiation problem relevance and the necessity of applying techniques to improve the reliability of embedded systems in critical applications. Although there are a plethora of methods in the literature to deal with the radiation effects, most of them have drawbacks mainly concerning in performance, area, and energy overhead. Even the proposed DCLS is not an exception. As the results showed, in some cases a high impact in performance is presented. Therefore, this issue must be fur-

ther investigated in order to find the best tradeoff that minimizes the performance losses for verification and recovery time.

As future work, the DCLS approach can be applied to other case study applications to assess the system better. Besides, the technique can be extended to other operating systems, such as Linux and eCos. Other optimizations in software partition and compilation can also be explored. In this work, all the processor's caches are disabled. However, this is not the most realistic scenario, mainly concerning the applications performance goals. Thus, an extension of the DCLS can be implemented to support the caches enabled. For that, during the checkpoint, data cache must be saved in the memory as well, while in the rollback operation the cache must be cleaned.

As the FPGA is particularly sensitive to soft errors, fault injection campaigns in the PL part of Zynq can be performed to analyze the Checker IP susceptibility and how it impacts in the DCLS effectiveness. Moreover, laser experiments can be conducted to assess the real vulnerability of the processor. Differently of heavy ions experiments, in which all the device is exposed, the laser is a controlled experiment. Regions of the DUT can be defined to be affected. Thus, only the processor can be selected to be hit, making possible evaluate the vulnerability without the FPGA interference.

The DCLS approach can be extended to protect heterogeneous multi-core processors embedded into MPSoCs. The vulnerability, performance, and power properties issues in heterogeneous multi-core processors can be explored. By the analysis of their susceptibility to soft errors, new techniques based on lockstep, and task redundancy and migration, can be developed to improve the system reliability. The cores in lockstep, applying the checkpoint and rollback methods combined with an external module for operation control is a robust solution to deal with both SDCs and SEFIs provoked by soft errors. Versions of lockstep can be exploited by implementing the approach in processor pairs, or with three or more redundancies. The challenge concerning to the overhead introduced by the technique mainly relates to performance and area. The interprocessor communication latency and the number of system verifications affect the performance directly. Furthermore, adding an extra module, besides the processor redundancy, can increase the susceptibility to the radiation effects. Thus, a full evaluation must be made to find the better lockstep version to improve the system resilience with minimal cost.

Some points shall be analyzed to find the best tradeoff:

- number of verifications performed during the application execution;
- number of lockstep-cores pairs;

- task scheduling to achieve the best performance;
- applying lockstep only on the critical tasks;
- fault latency investigation.

Through studying the relation between the heterogeneous multi-core processor vulnerability, task criticality, and execution time, an efficient solution to fault mitigation with fewer penalties can be achieved. Therefore, future research can contribute to alleviating the dependability problem in embedded multi-core processors by offering a fault tolerant solution with minimal drawbacks.

REFERENCES

ABATE, F.; STERPONE, L.; VIOLANTE, M. A new mitigation approach for soft errors in embedded processors. **IEEE Transactions on Nuclear Science**, v. 55, n. 4, p. 2063–2069, Aug 2008. ISSN 0018-9499.

AGUIAR, V. et al. Experimental setup for single event effects at the são paulo 8ud pelletron accelerator. **Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms**, v. 332, p. 397–400, 2014. ISSN 0168-583X. 21st International Conference on Ion Beam Analysis.

ALTERA. **Cyclone V SoC Development Board Reference Manual**. [S.l.], 2015.

ARM. **Cortex-A9 Technical Reference Manual. Revision: r2p2**. [S.l.], 2010.

ARM. **Cortex-R5 and Cortex-R5F Technical Reference Manual. Rev:r1p1**. [S.l.], 2011.

ARM. **ARM® Architecture Reference Manual. ARMv7-A and ARMv7-R edition**. [S.l.], 2012.

ARM. **ARM® Compiler armcc User Guide. Version 5.05. DUI0472K**. [S.l.], 2014.

AVIZIENIS, A. et al. Basic concepts and taxonomy of dependable and secure computing. **IEEE Transactions on Dependable and Secure Computing**, v. 1, n. 1, p. 11–33, Jan 2004. ISSN 1545-5971.

AVNET. **ZedBoard Getting Started Guide. Version 7.0**. [S.l.], 2017.

AZAMBUJA, J. R. et al. Heta: Hybrid error-detection technique using assertions. **IEEE Transactions on Nuclear Science**, v. 60, n. 4, p. 2805–2812, Aug 2013. ISSN 0018-9499.

AZAMBUJA, J. R.; KASTENSMIDT, F.; BECKER, J. **Hybrid Fault Tolerance Techniques to Detect Transient Faults in Embedded Processors**. [S.l.: s.n.], 2014. ISSN 1467-9280. ISBN 9780874216561.

BAHARVAND, F.; MIREMADI, S. G. Lexact: Low energy n-modular redundancy using approximate computing for real-time multicore processors. **IEEE Transactions on Emerging Topics in Computing**, PP, n. 99, p. 1–1, 2017.

BARRY, R. **FreeRTOS**. 2017. [Accessed September-2017]. Available from Internet: <http://www.freertos.org>.

BAUMANN, R. C. Radiation-induced soft errors in advanced semiconductor technologies. **IEEE Transactions on Device and Materials Reliability**, v. 5, n. 3, p. 305–316, Sept 2005. ISSN 1530-4388.

BOWEN, N. S.; PRADHAM, D. K. Processor- and memory-based checkpoint and rollback recovery. **Computer**, v. 26, n. 2, p. 22–31, Feb 1993. ISSN 0018-9162.

CHIELLE, E. et al. Hybrid soft error mitigation techniques for cots processor-based systems. In: **2016 17th Latin-American Test Symposium (LATS)**. [S.l.: s.n.], 2016. p. 99–104.

CHIELLE, E. et al. S-seta: Selective software-only error-detection technique using assertions. **IEEE Transactions on Nuclear Science**, v. 62, n. 6, p. 3088–3095, Dec 2015. ISSN 0018-9499.

DODD, P. E. et al. Neutron-induced latchup in srams at ground level. In: **2003 IEEE International Reliability Physics Symposium Proceedings, 2003. 41st Annual.** [S.l.: s.n.], 2003. p. 51–55.

DOMEIKA, M. **Software Development for Embedded Multi-core Systems. A Practical Guide Using Embedded Intel ® Architecture.** [S.l.]: Elsevier Inc., 2008. ISBN 978-0-7506-8539-9.

DUBROVA, E. **Fault Tolerant Design: An Introduction**. 2008. [Accessed September-2017]. Available from Internet: <http://www.pld.ttu.ee/IAF0530/draft.pdf>.

ENTRENA, L. et al. Soft error sensitivity evaluation of microprocessors by multilevel emulation-based fault injection. **IEEE Transactions on Computers**, v. 61, n. 3, p. 313–322, March 2012. ISSN 0018-9340.

ESA. **ESA/SCC Basic specification n. 25100: Single Event Effects Test Method and Guidelines**. Noordwijk, Netherlands, 2005.

FAYYAZ, M.; VLADIMIROVA, T. Fault-tolerant distributed approach to satellite on-board computer design. In: **2014 IEEE Aerospace Conference**. [S.l.: s.n.], 2014. p. 1–12. ISSN 1095-323X.

GINOSAR, R. Survey of processors for space. In: **DASIA**. [S.l.: s.n.], 2012. p. 1–5.

GOLOUBEVA, O. et al. **Software-implemented hardware fault tolerance**. [S.l.]: Springer Science & Business Media, 2006.

GOMAA, M. A. et al. Transient-fault recovery for chip multiprocessors. **IEEE Micro**, v. 23, n. 6, p. 76–83, Nov 2003. ISSN 0272-1732.

GOMEZ-CORNEJO, J. et al. Fast context reloading lockstep approach for seus mitigation in a fpga soft core processor. In: **IECON 2013 - 39th Annual Conference of the IEEE Industrial Electronics Society**. [S.l.: s.n.], 2013. p. 2261–2266. ISSN 1553-572X.

HAGEN, W. von. **The Definitive Guide to GCC**. 2. ed. [S.l.]: Apress, 2006. ISBN 978-1-59059-585-5.

HAN, J.; ORSHANSKY, M. Approximate computing: An emerging paradigm for energy-efficient design. In: **2013 18th IEEE European Test Symposium (ETS)**. [S.l.: s.n.], 2013. p. 1–6. ISSN 1530-1877.

JOHNSON, O.; DINYO, O. Comparative analysis of single-core and multi-core systems. v. 7, p. 117–130, 12 2015.

LAFRIEDA, C. et al. Utilizing dynamically coupled cores to form a resilient chip multiprocessor. In: **37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)**. [S.l.: s.n.], 2007. p. 317–326. ISSN 1530-0889.

LINS, F. M. **The effects of the compiler optimizations in embedded processors reliability**. Thesis (PhD) — Universidade Federal do Rio Grande do Sul, Instituto de Informática, Programa de Pós-Graduação em Microeletrônica, Brazil, 2017.

LINS, F. M. et al. Register file criticality and compiler optimization effects on embedded microprocessor reliability. **IEEE Transactions on Nuclear Science**, v. 64, n. 8, p. 2179–2187, Aug 2017. ISSN 0018-9499.

MAHMOOD, A.; MCCLUSKEY, E. J. Concurrent error detection using watchdog processors-a survey. **IEEE Transactions on Computers**, v. 37, n. 2, p. 160–174, Feb 1988. ISSN 0018-9340.

MANIATAKOS, M. et al. Instruction-level impact analysis of low-level faults in a modern microprocessor controller. **IEEE Transactions on Computers**, v. 60, n. 9, p. 1260–1273, Sept 2011. ISSN 0018-9340.

MEDINA, N. H. et al. Experimental Setups for Single Event Effect Studies. **Journal of Nuclear Physics, Material Sciences, Radiation and Applications**, v. 4, n. 1, p. 13–23, Aug 2016.

MICROCHIP. **Rad Hard Processors**. 2017. [Accessed September-2017]. Available from Internet: <http://www.microchip.com/design-centers/rad-hard/processors>.

MONDRAGON, A. F. **AC 2012-4835: Hard Core Vs. Soft Core: A Debate**. 2012. [Accessed September-2017]. Available from Internet: <https://www.researchgate.net/profile/Antonio_Mondragon-Torres/publication/236844584_Hard_Core_vs_Soft_Core_A_Debate/links/004635195c82d1354f000000/Hard-Core-vs-Soft-Core-A-Debate.pdf>.

MOYER, W.; ROCHFORD, M.; SANTO, D. **Error detection and communication of an error location in multi-processor data processing system having processors operating in Lockstep**. Google Patents, 2012. US Patent 8,090,984. [Accessed September-2017]. Available from Internet: <http://www.google.ch/patents/US8090984>.

MUKHERJEE, S. S.; KONTZ, M.; REINHARDT, S. K. Detailed design and evaluation of redundant multi-threading alternatives. In: **Proceedings 29th Annual International Symposium on Computer Architecture**. [S.l.: s.n.], 2002. p. 99–110. ISSN 1063-6897.

NAZAR, G. L.; CARRO, L. Fast single-fpga fault injection platform. In: **2012 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)**. [S.l.: s.n.], 2012. p. 152–157. ISSN 1550-5774.

NORMAND, E. Correlation of inflight neutron dosimeter and seu measurements with atmospheric neutron model. **IEEE Transactions on Nuclear Science**, v. 48, n. 6, p. 1996–2003, Dec 2001. ISSN 0018-9499.

OLIVEIRA, Á. B. de et al. Analyzing the impact of software optimizations in lockstep dual-core arm a9 under heavy ion induced soft errors. In: **European Conference on Radiation and Its Effects on Components and Systems (RADECS)**. [S.l.: s.n.], 2017. p. 1–4. [To be published].

OLIVEIRA, Á. B. de; TAMBARA, L. A.; KASTENSMIDT, F. L. Applying lockstep in dual-core arm cortex-a9 to mitigate radiation-induced soft errors. In:

2017 IEEE 8th Latin American Symposium on Circuits Systems (LASCAS). [s.n.], 2017. p. 1–4. [Accessed September-2017]. Available from Internet: <http://dx.doi.org/10.1109/LASCAS.2017.7948063>.

OLIVEIRA, Á. B. de; TAMBARA, L. A.; KASTENSMIDT, F. L. Exploring performance overhead versus soft error detection in lockstep dual-core arm cortex-a9 processor embedded into xilinx zynq apsoc. In: ____. **Applied Reconfigurable Computing: 13th International Symposium, ARC 2017, Delft, The Netherlands, April 3-7, 2017, Proceedings**. Cham: Springer International Publishing, 2017. p. 189–201. ISBN 978-3-319-56258-2. [Accessed September-2017]. Available from Internet: <http://dx.doi.org/10.1007/978-3-319-56258-2_17>.

OSINSKI, L.; LANGER, T.; MOTTOK, J. A survey of fault tolerance approaches on different architecture levels. In: **ARCS 2017; 30th International Conference on Architecture of Computing Systems**. [S.l.: s.n.], 2017. p. 1–9.

PHAM, H. M.; PILLEMENT, S.; PIESTRAK, S. J. Low-overhead fault-tolerance technique for a dynamically reconfigurable softcore processor. **IEEE Transactions on Computers**, v. 62, n. 6, p. 1179–1192, June 2013. ISSN 0018-9340.

QUINN, H. Challenges in testing complex systems. **IEEE Transactions on Nuclear Science**, v. 61, n. 2, p. 766–786, April 2014. ISSN 0018-9499.

RECH, P. et al. Impact of gpus parallelism management on safety-critical and hpc applications reliability. In: **2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks**. [S.l.: s.n.], 2014. p. 455–466. ISSN 1530-0889.

REINHARDT, S. K.; MUKHERJEE, S. S. Transient fault detection via simultaneous multithreading. In: **Proceedings of 27th International Symposium on Computer Architecture (IEEE Cat. No.RS00201)**. [S.l.: s.n.], 2000. p. 25–36. ISSN 1063-6897.

REORDA, M. S. et al. A low-cost see mitigation solution for soft-processors embedded in systems on pogrammable chips. In: **2009 Design, Automation Test in Europe Conference Exhibition**. [S.l.: s.n.], 2009. p. 352–357. ISSN 1530-1591.

RODRIGUES, G. S.; KASTENSMIDT, F. L. Evaluating the behavior of successive approximation algorithms under soft errors. In: **2017 18th IEEE Latin American Test Symposium (LATS)**. [S.l.: s.n.], 2017. p. 1–6.

RODRIGUES, G. S. et al. Analyzing the impact of fault-tolerance methods in arm processors under soft errors running linux and parallelization apis. **IEEE Transactions on Nuclear Science**, v. 64, n. 8, p. 2196–2203, Aug 2017. ISSN 0018-9499.

ROSSI, D. et al. Multiple transient faults in logic: an issue for next generation ics? In: **20th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'05)**. [S.l.: s.n.], 2005. p. 352–360. ISSN 1550-5774.

SALEHI, M.; EJLALI, A.; AL-HASHIMI, B. M. Two-phase low-energy n-modular redundancy for hard real-time multi-core systems. **IEEE Transactions on Parallel and Distributed Systems**, v. 27, n. 5, p. 1497–1510, May 2016. ISSN 1045-9219.

SALEHI, M. et al. Drvs: Power-efficient reliability management through dynamic redundancy and voltage scaling under variations. In: **2015 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)**. [S.l.: s.n.], 2015. p. 225–230.

SIEGLE, F. et al. Mitigation of radiation effects in sram-based fpgas for space applications. **ACM Comput. Surv.**, ACM, New York, NY, USA, v. 47, n. 2, p. 37:1–37:34, jan. 2015. ISSN 0360-0300. [Accessed September-2017]. Available from Internet: <http://doi.acm.org/10.1145/2671181>.

SMOLENS, J. C. et al. Reunion: Complexity-effective multicore redundancy. In: **2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)**. [S.l.: s.n.], 2006. p. 223–234. ISSN 1072-4451.

STALLMAN, R. M.; COMMUNITY the G. D. **Using the GNU Compiler Collection: For gcc version 4.9.1**. [S.l.], 2014.

TAMBARA, L. A. **Analyzing the Impact of Radiation-induced Failures in All Programmable System-on-Chip Devices**. Thesis (PhD) — Universidade Federal do Rio Grande do Sul, Instituto de Informática, Programa de Pós-Graduação em Microeletrônica, Brazil, 2017.

TAMBARA, L. A. et al. Heavy ions induced single event upsets testing of the 28 nm xilinx zynq-7000 all programmable soc. In: **2015 IEEE Radiation Effects Data Workshop (REDW)**. [S.l.: s.n.], 2015. p. 1–6.

TAMBARA, L. A. et al. Analyzing the impact of radiation-induced failures in programmable socs. **IEEE Transactions on Nuclear Science**, v. 63, n. 4, p. 2217–2224, Aug 2016. ISSN 0018-9499.

TAYLOR, A. How to use interrupts on the zynq soc. **Xcell Journal**, p. 38–43, 2014.

TROPPMANN, R.; FUESSL, B. **Delayed lock-step cpu compare**. Google Patents, 2008. US Patent App. 12/042,080. [Accessed September-2017]. Available from Internet: <https://www.google.com/patents/US20080244305>.

VELAZCO, R.; FAURE, F. Error rate prediction of digital architectures: Test methodology and tools. In: ____. **Radiation Effects on Embedded Systems**. Dordrecht: Springer Netherlands, 2007. p. 233–258. ISBN 978-1-4020-5646-8. [Accessed September-2017]. Available from Internet: <https://doi.org/10.1007/978-1-4020-5646-8_11>.

VELAZCO, R.; REZGUI, S.; ECOFFET, R. Predicting error rate for microprocessor-based digital architectures through c.e.u. (code emulating upsets) injection. **IEEE Transactions on Nuclear Science**, v. 47, n. 6, p. 2405–2411, Dec 2000. ISSN 0018-9499.

VIJAYKUMAR, T. N.; POMERANZ, I.; CHENG, K. Transient-fault recovery using simultaneous multithreading. In: **Proceedings 29th Annual International Symposium on Computer Architecture**. [S.l.: s.n.], 2002. p. 87–98. ISSN 1063-6897.

VIOLANTE, M. et al. A low-cost solution for deploying processor cores in harsh environments. **IEEE Transactions on Industrial Electronics**, v. 58, n. 7, p. 2617–2626, July 2011. ISSN 0278-0046.

VORAGO. **VA10820 - Radiation Hardened ARM® Cortex®-M0 MCU**. 2017. [Accessed September-2017]. Available from Internet: <http://www.voragotech.com/products/VA10820>.

WANG, C. et al. Compiler-managed software-based redundant multi-threading for transient fault detection. In: **International Symposium on Code Generation and Optimization (CGO'07)**. [S.l.: s.n.], 2007. p. 244–258.

XILINX. **7 Series FPGAs Configuration User Guide UG470 (v1.11)**. [S.l.], 2016.

XILINX. **Zynq-7000 All Programmable SoC Technical Reference Manual UG585 (v1.11)**. [S.l.], 2016.

XILINX. **MicroBlaze Processor Reference Guide UG984 (v2017.2)**. [S.l.], 2017.

XILINX. **Zynq® UltraScale+™ MPSoCs Product Tables and Product Select Guide**. [S.l.], 2017.

APPENDIX A — PUBLICATIONS

First author:

1. **Exploring Performance Overhead Versus Soft Error Detection in Lockstep Dual-Core ARM Cortex-A9 Processor Embedded into Xilinx Zynq APSoC.** Á. B. de Oliveira, L. A. Tambara, F. L. Kastensmidt. Lecture Notes in Computer Science. 1ed.: Springer International Publishing, 2017, v. 10216, pp. 189-201. doi: 10.1007/978-3-319-56258-2_17.
2. **Applying Lockstep in Dual-Core ARM Cortex-A9 to Mitigate Radiation-Induced Soft Errors.** Á. B. de Oliveira, L. A. Tambara, F. L. Kastensmidt. 2017 IEEE 8th Latin American Symposium on Circuits Systems (LASCAS), Feb 2017, pp. 1–4. doi: 10.1109/LASCAS.2017.7948063.
3. **Analyzing Lockstep Dual-Core ARM Cortex-A9 Soft Error Mitigation in FreeRTOS Applications.** Á. B. de Oliveira, G. S. Rodrigues, F. L. Kastensmidt. 2017 30th Symposium on Integrated Circuits and Systems Design (SBCCI), Fortaleza, 2017, pp. 84-89.
4. **Analyzing the Impact of Software Optimizations in Lockstep Dual-Core ARM A9 under Heavy Ion Induced Soft Errors.** Á. B. de Oliveira, G. S. Rodrigues, F. L. Kastensmidt, N. Added, E. L. A. Macchione, V. A. P. Aguiar, N. H. Medina, M. A. G. Silveira. European Conference on Radiation and Its Effects on Components and Systems (RADECS), 2017. (To be published)

Co-author:

1. **Analyzing the Impact of Fault Tolerance Methods in ARM Processors under Soft Errors running Linux and Parallelization APIs.** G. S. Rodrigues, F. Rosa, Á. B. de Oliveira, F. L. Kastensmidt, L. Ost and R. Reis, IEEE Transactions on Nuclear Science, vol. 64, no. 8, pp. 2196-2203, Aug. 2017. doi: 10.1109/TNS.2017.2706519.