Single Event Upset Mitigation Techniques in Reconfigurable Hardware

Michail Vavouras

Supervisor: Dr. Christos-Savvas Bouganis

Submitted in part fulfilment of the requirements for the degree of Doctor of Philosophy of Imperial College London

September 2017
Abstract

Advances in semiconductor technology using smaller sizes of transistors in order to fit more of them in the same area and increase performance, pose a threat for the reliability of integrated circuits. Technology scaling accelerates transistor ageing and degradation, causing more faults during the lifetime of an integrated circuit. Sources of faults such as manufacturing defects, degradation and ageing of transistors degrade the performance of integrated circuits leading to faults with a permanent effect that might be catastrophic for certain applications. A special case of integrated circuits, FPGAs, suffer from radiation-induced faults since they contain million of bits for the configuration of their resources that if flipped due to radiation might change the intended functionality of the application running on the FPGA, causing a failure. However, FPGAs can be dynamically reconfigured in the field and mitigate radiation effects providing fault-tolerance and high availability.

A novel fault-tolerant architecture for an artificial pancreas application is proposed that consists of a mixed substrate of ASIC and FPGA. Fault detection is provided through modular redundancy, and dynamic reconfiguration is used as a repair mechanism. Experimental results show that 5,100x lower Probability of Failures per Hour (PFH) than a Dual Modular Redundancy (DMR) for permanent faults can be achieved with 2.4x more area than DMR. In addition, the proposed solution achieves 83x lower PFH than a Triple Modular Redundancy (TMR) with 1.6x area overheads when considering transient faults.

A framework supporting fault injection at the configuration memory of an SRAM FPGA and scrubbing was developed throughout this work. The framework supports various SEU and scrub rates and is implemented on the modern ZYNQ FPGA architecture. Existing scrubbing strategies were implemented for a second-order polynomial case study together with two new scrubbing techniques taking into consideration area information of the modules of the application. Experimental results show that the area-driven scrubbing technique achieves 43.6% LUTs and 40.9% REGs savings when compared to a DMR design. The area-driven technique for the partial TMR design saves 15% LUTs and 23% REGs area as compared to the TMR without sacrificing availability, but with increased power consumption for scrubbing.

The conclusion of the work is that dynamic reconfiguration techniques can be effectively applied in FPGAs for trading-off resources and power consumption for availability.
Acknowledgements

First of all, I would like to thank my supervisor Dr. Christos Bouganis for his valuable guidance and support for this work. Christos gave me all the freedom to perform my research and was always there for discussing the results of this thesis. I have to note here that without Christos passion and motivation, this work would not have been done.

I am grateful to Mrs Wiesia Hsissen, the CAS group administrator for advising me and helping me with the college rules, policies and deadlines.

I would like to thank the members of the Reliability Club, a weekly group meeting with Professor George Constantinides, Dr Ed Stott, Dr. Josh Levine, Dr. James Davies and Dr. Rui Duarte for their suggestions and guidance on the progression of this work.

Special thanks go to my colleagues Dr. Grigorios Mingas, Dr. Andrea Suardi, Shane Flemming and the rest of the CAS group members for being helpful, kind and generous to me, allowing me to complete this thesis.

Moreover, I would like to thank Dr. Athanasios Gkelias, Alex, Kwstas, Orestis, Akis, Fran and Marianna, Rigas and Panos for their support and for being very good friends.

Last but not least I would like to thank my parents, my sister and my brother for everything they have done and continue to do for me.

This work is partially funded by the European FP7 DeSyRe Project.
Dedication

To my parents.
Declaration

The material contained within this thesis has not previously been submitted for a degree at the Imperial College London or any other university. The work reported has been conducted by the author unless indicated otherwise.

Copyright Declaration

The copyright of this thesis rests with the author and is made available under a Creative Commons Attribution Non-Commercial No Derivatives licence. Researchers are free to copy, distribute or transmit the thesis on the condition that they attribute it, that they do not use it for commercial purposes and that they do not alter, transform or build upon it. For any reuse or redistribution, researchers must make clear to others the licence terms of this work.
# Contents

Abstract i

Acknowledgements iii

1 Introduction 1

1.1 Contributions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
1.2 Publications . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
1.3 Outline . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

2 Background 5

2.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
2.2 Faults, Errors and Failures . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
2.3 Types of Faults . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
2.4 Sources of Faults . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
  2.4.1 Ageing and Degradation . . . . . . . . . . . . . . . . . . . . . . . . . . 6
  2.4.2 Manufacturing Defects . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
  2.4.3 Radiation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
2.5 Fault Detection . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
  2.5.1 Spatial Redundancy . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
  2.5.2 Concurrent Error Detection . . . . . . . . . . . . . . . . . . . . . . . . 11
  2.5.3 Off-line Testing . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
2.5.4 On-Line Testing ........................................... 14
2.5.5 Comparison of Fault Detection Techniques .......... 15
2.6 Fault Recovery in FPGAs ................................... 17
  2.6.1 Permanent Fault Recovery .............................. 17
  2.6.2 Transient Fault Recovery ............................... 24
2.7 SEU Mitigation Frameworks ................................. 28
2.8 Conclusion .................................................. 30

3 A Hybrid ASIC/FPGA Fault-Tolerant Artificial Pancreas 31
  3.1 Introduction ................................................ 31
  3.2 Artificial Pancreas ......................................... 31
  3.3 Baseline Artificial Pancreas ............................... 34
  3.4 Fault Model and Safety Requirements .................... 37
  3.5 Generic description of the fault-tolerant technique .... 38
  3.6 Proposed Artificial Pancreas .............................. 43
    3.6.1 The Hybrid-Substrate Fabric ......................... 44
    3.6.2 Fault detection ....................................... 47
    3.6.3 Fault Correction ...................................... 49
  3.7 Evaluation Results ........................................ 50
    3.7.1 Target technology node ............................... 51
    3.7.2 Permanent Fault Evaluation ......................... 51
    3.7.3 Transient Fault Evaluation ......................... 53
  3.8 Conclusion ................................................ 55

4 FPGA Platform for Investigating the Impact of SEUs to a Design 56
  4.1 Introduction ................................................ 56
  4.2 High level System Overview .............................. 57
    4.2.1 Xilinx Zedboard ...................................... 57
4.2.2 Xilinx ZYNQ FPGA Architecture ........................................... 58
4.2.3 Hardware ............................................................................ 58
4.2.4 Configuration Memory .......................................................... 59
4.2.5 High level view of fault injection and scrubbing ...................... 62
4.2.6 AXI Timers ......................................................................... 65
4.2.7 Hardware Support for the interrupts ...................................... 65
4.2.8 Modifications at the hardware level ...................................... 66
4.2.9 Software .............................................................................. 69
4.3 SEU Framework Performance Evaluation ................................. 72
4.4 Conclusion ............................................................................ 80

5 Area-Driven Partial Reconfiguration for SEU Mitigation ............... 82
  5.1 Introduction ........................................................................... 82
  5.2 Baseline .................................................................................. 84
  5.3 DMR system ........................................................................... 84
  5.4 Proposed SEU Mitigation Technique for a partial DMR system .... 84
  5.5 Formal description of the method ............................................ 88
    5.5.1 Blind scrubbing on the baseline design .............................. 88
    5.5.2 On-demand scrubbing on the DMR design ....................... 88
    5.5.3 Proposed techniques on the reconfigurable DMR design ...... 90
  5.6 Second-Order Polynomial Case Study .................................... 94
  5.7 High level system operation .................................................... 96
  5.8 Performance Evaluation of FPGA scrubbing techniques (DMR) ... 97
    5.8.1 SEU rate ........................................................................ 98
    5.8.2 Availability .................................................................... 102
    5.8.3 Area Resources ............................................................... 105
    5.8.4 Mean Time to Repair (MTTR) .......................................... 107
List of Tables

2.1 Comparison of fault detection techniques [54] .......................... 16
2.2 Comparison of permanent fault recovery techniques .................. 25
2.3 Comparison of SEU mitigation techniques and frameworks ............ 29

3.1 Breakdown of resources of the ASIC part of the proposed architecture ... 53
3.2 Hardware and Timing Dependability metrics for permanent faults of the proposed and DMR versions of the pancreas application ............... 53
3.3 Hardware and Timing Dependability metrics for transient faults of the proposed and TMR versions of the pancreas application ............... 54

4.1 Interrupts description for the SEU mitigation framework ................ 71
4.2 Resources utilization of the framework on the ZYNQ without the CUT ... 74
4.3 Mean and standard deviation values for various fscrub/fseu ratios .......... 80

5.1 SEU rates and number of essential bits for Baseline, DMR, uniform and area-driven scrubbing techniques ................................. 98
5.2 Number of essential bits for the ADD, MULT1 and MULT2 modules ........ 99
5.3 fscrub/fseu, SEU and scrubbing rates for the round robin scheduling and area-driven techniques ................................................. 102
5.4 Resources utilisation of the three designs .................................. 106
5.5 MTTR for the baseline, the DMR and the Reconfigurable DMR designs ... 109
5.6 Bitstream size in bytes for the three designs ............................... 109
5.7 SEU rates and size of essential bits for the on-demand scrubbing (TMR) and round robin scheduling and area-driven scrubbing techniques (Reconfigurable TMR) ......................................................... 127

5.8 fscrub/fseu, SEU and scrubbing rates for round robin scheduling and area-driven techniques of the Reconfigurable TMR design ............................... 130

5.9 Resources utilisation of the TMR and Reconfigurable TMR designs ................. 133

5.10 MTTR for TMR and Reconfigurable TMR designs .................................... 135

5.11 Bitstream size in bytes for the three designs ........................................... 136
List of Figures

2.1 Duplication with comparison ........................................... 10
2.2 General architecture of a concurrent error detection scheme [43] ......... 12
2.3 Built-in-self-test structure for FPGAs [49] .................................. 14
2.4 Roving area under test [51] .................................................. 14
2.5 FPGA configuration (a) before and (b) after a defect is tolerated [56] .... 18
2.6 Self-repair architecture [58] .................................................. 18
2.7 Block structured architecture [59] .......................................... 19
2.8 Fault tolerant segment method [60] ........................................ 19
2.9 Fault tolerant grid method [60] .............................................. 20
2.10 SRAM shifting (a) king allocation and (b) horse allocation [61] .......... 20
2.11 SRAM shifting example [61] ................................................ 21
2.12 Pebble shifting (a) a placed circuit with a faulty resource; (b) the initial
graph created from the circuit; and (c) the reconfigured circuit with thick
grey lines representing the shifted paths [62] .................................. 22
2.13 Tiling (a) one PLB in the top left tile is faulty; (b) the reconfigured circuit
that bypasses the faulty PLB [65] ................................................. 23
2.14 Internal vs External scrubber [71] .......................................... 26
3.1 Device for continous glucose sensing ..................................... 32
3.2 State-of-the-art artificial pancreas [79] .................................... 33
3.3 Proposed artificial pancreas implant [7] ..................................... 34
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.4</td>
<td>Block Diagram of the Baseline Artificial Pancreas</td>
<td>35</td>
</tr>
<tr>
<td>3.5</td>
<td>Block diagram of a modular design</td>
<td>38</td>
</tr>
<tr>
<td>3.6</td>
<td>Block diagram of a proposed fault tolerant design</td>
<td>39</td>
</tr>
<tr>
<td>3.7</td>
<td>FPGA-based system prototype</td>
<td>44</td>
</tr>
<tr>
<td>3.8</td>
<td>Proposed artificial pancreas</td>
<td>46</td>
</tr>
<tr>
<td>3.9</td>
<td>Off-line testing of the arithmetic units of the proposed artificial pancreas</td>
<td>48</td>
</tr>
<tr>
<td>3.10</td>
<td>Component Reconfiguration</td>
<td>50</td>
</tr>
<tr>
<td>4.1</td>
<td>Zedboard</td>
<td>58</td>
</tr>
<tr>
<td>4.2</td>
<td>Hardware prototype</td>
<td>59</td>
</tr>
<tr>
<td>4.3</td>
<td>Venn diagram of the relationship between device configuration bits, essential bits, prioritised essential bits and critical bits [91]</td>
<td>61</td>
</tr>
<tr>
<td>4.4</td>
<td>Design example block diagram of the SEM Core</td>
<td>63</td>
</tr>
<tr>
<td>4.5</td>
<td>Block diagram of the AXI HWICAP [93]</td>
<td>64</td>
</tr>
<tr>
<td>4.6</td>
<td>Arbitration of the ICAP primitive between the SEM Core and the HWICAP</td>
<td>67</td>
</tr>
<tr>
<td>4.7</td>
<td>Fault injection process using the SEM Core</td>
<td>68</td>
</tr>
<tr>
<td>4.8</td>
<td>Control logic for sharing the ICAP between the SEM core and the HWICAP module</td>
<td>69</td>
</tr>
<tr>
<td>4.9</td>
<td>Flowchart of the software</td>
<td>70</td>
</tr>
<tr>
<td>4.10</td>
<td>Performance of the framework</td>
<td>73</td>
</tr>
<tr>
<td>4.11</td>
<td>Error distribution of actual vs theoretical fault injection for fscrub/fseu = 1/16</td>
<td>75</td>
</tr>
<tr>
<td>4.12</td>
<td>Error distribution of actual vs theoretical fault injection for fscrub/fseu = 1/8</td>
<td>76</td>
</tr>
<tr>
<td>4.13</td>
<td>Error distribution of actual vs theoretical fault injection for fscrub/fseu = 1/4</td>
<td>76</td>
</tr>
<tr>
<td>4.14</td>
<td>Error distribution of actual vs theoretical fault injection for fscrub/fseu = 1/2</td>
<td>77</td>
</tr>
</tbody>
</table>
List of Abbreviations

AES  Advanced Encryption Standard

ASIC  Application-Specific Integrated Circuit

AXI  Advanced eXtensible Interface

BIST  Built-In Self Tests

BLE  Basic Logic Element

BRAM  Block Random Access Memory

CED  Concurrent Error Detection

CLB  Configurable Logic Block

CPU  Central Processing Unit

CRC  Cyclically Redundancy Check

CUT  Circuit Under Test

COTS  Commercial Off The Shelf

DC  Diagnostic Coverage

DDR3  Double Data Rate Type 3

DMR  Dual Modular Redundancy

DSP  Digital Signal Processing
DTI Detection Time Interval

DWC Duplication With Comparison

ECC Error Correcting Codes

EDAC Error Detection and Correction Codes

FF Flip Flop

FIR Finite Impulse Response

FMEDA Failure Modes, Effects and Diagnostic Analysis

FPGA Field-Programmable Gate Array

FIT Failures in Time

FSM Finite State Machine

GEO Geostationary Earth Orbit

GSR Galactic Cosmic Ray

HCI Hot Carrier Injection

HFT-of-N Hardware Fault Tolerance of N

HWICAP Hardware Internal Configuration Access Port

ICAP Internal Configuration Access Port

JTAG Joint Test Action Group

LED Light Emitting Diode

LEO Low Earth Orbit

LFSR Linear Feedback Shift Register

LSB Least Significant Bit
LUT  Lookup Table

MOSFET  Metal-Oxide Semiconductor Field-Effect Transistor

MRT  Mean Repair Time

MSB  Most Significant Bit

MTTD  Mean Time to Detect

MTTR  Mean Time to Repair

MUX  Multiplexer

NBTI  Negative Bias Temperature Instability

NMOS  N-Channel Metal-Oxide Semiconductor

PCAP  Processor Configuration Access Port

PBTI  Positive Bias Temperature Instability

PFH  Probability of Failures per Hour

PL  Programmable Logic

PLB  Programmable Logic Block

PLL  Phase-Locked Loop

PMOS  P-Channel Metal-Oxide Semiconductor

PS  Processing System

PST  Process Safety Time

REG  Register

RPR  Reduced Precision Redundancy

RTL  Register Transfer Level
**SECDED** Single Error Correction and Double Error Detection

**SEE** Single Event Effects

**SEL** Single Event Latchup

**SEM** Soft Error Mitigation

**SET** Single Event Transient

**SEU** Single Event Upset

**SEFI** Single Event Functional Interrupt

**SFF** Safe Failure Fraction

**SIL** Safety Integrity Level

**SRAM** Static Random Access Memory

**SSF** Single Stuck-at-Fault

**STAR** Self Testing Areas

**TID** Total Ionizing Dose

**TDDB** Time-Dependent Gate Oxide Breakdown

**TMR** Triple Modular Redundancy

**VHDL** Very High Speed Integrated Circuit Hardware Description Language

**WCET** Worst Case Execution Time
Chapter 1

Introduction

Advances in semiconductor technology using smaller sizes of transistors in order to fit more of them in the same area and increase performance, pose a threat for the reliability of integrated circuits [1]. Technology scaling accelerates transistor ageing and degradation, causing more faults during the lifetime of an integrated circuit [2]. For instance, by the 16 nm generation, the failure rate will be almost 100 times that at 180 nm [3]. Transistor ageing is expected to be a significant source of errors in technologies beyond 16 nm and will reduce the lifetime of semiconductor devices [4].

Biomedical companies use significantly old and mature semiconductor technologies to manufacture reliable systems. However, relying on older technology nodes has an excessive performance and energy cost compared to using recent emerging technologies. Such cost is often prohibitive for future medical embedded systems [5]. Safety critical applications such as an artificial pancreas are vulnerable to faults, and more aggressive design techniques should be used to tolerate more faults and extend their yield and lifetime.

Recently, the use of FPGAs as implementation platforms for various applications as compared to Application-Specific Integrated Circuits (ASICs) has grown, as they offer low non-recurring engineering costs and fast prototyping. Nowadays, Field-Programmable Gate Arrays (FPGAs) have a significant resource and performance overhead compared to ASICs, however, this gap tends to decrease as more building blocks such as SRAMs, adders and multipliers have dedicated hardware support [6]. The major advantage of FPGAs as
compared to ASICs is that they can be reconfigured in the field offering flexibility in applications that require fault-tolerance [7].

However, modern SRAM-based FPGAs have many configuration bits controlling LUTs, flip-flops, routing, DSPs and BRAMs. More than 90% of the configuration bitsream of an SRAM-based FPGA belongs to routing [8]. Xilinx performed real-time experiments in various locations and altitudes for many device families and technology nodes. The real time soft error rate caused by protons and neutrons for a ZYNQ FPGA device at 28nm was 72 Failures in Time (FIT) per Mb for configuration memory and 66 FIT per Mb for BRAMs [9]. Thus, a primary focus should be given to the vulnerability due to Single Event Upsets (SEU) in configuration memory of the FPGAs. SEUs caused by radiation particles such as neutrons and protons are not only witnessed in space but also on earth, and in particular in high altitude areas and scientific laboratories such as CERN [9]. Depending on the nature of the application, a strike of a particle on a semiconductor node might cause an SEU that will affect the functionality of the application leading to a system failure. Due to the type of the targeted device (i.e FPGA), SEUs can be removed by reconfiguration, while faults that are destructive for an SRAM FPGA, demand relocation of the application, if possible, on another functional part of the device [10].

Availability, defined as the percentage of time a system operates correctly over the total time of operation, is a design objective for electronic systems. FPGA-based fault-tolerant systems that require high availability, use redundancy schemes such as Dual Modular Redundancy (DMR) and Triple Modular Redundancy (TMR) to detect and mask faults at the application level and scrubbing, that is the rewriting of the configuration memory of the FPGA device with an error-free bitstream, for fault correction. When high availability is demanded and power consumption and area is of importance, novel techniques should be investigated for both detection and correction of faults.

1.1 Contributions

The contributions of this work are:
• A Hybrid ASIC/FPGA architecture for providing fault-tolerance to an artificial pancreas and extend its lifetime at the cost of low area overheads compared to standard approaches based on DMR and TMR. Failure Modes, Effects and Diagnostic Analysis (FMEDA) is performed for permanent and transient faults for the artificial pancreas (Chapter 3).

• The design and implementation of a new SEU mitigation framework supporting fault injection at the SRAM configuration layer and FPGA scrubbing. The framework operates on a single FPGA device (ZYNQ) and uses dynamic reconfiguration techniques for error detection and correction, leading to low-cost solutions (Chapter 4).

• A new hardware architecture (partial DMR) with a novel scrubbing technique that takes into account area information and saves resources as compared to a DMR with on-demand scrubbing for a second order polynomial case study. Investigation of the trade-off between availability, area and power of blind scrubbing, on-demand scrubbing and the area-driven scrubbing technique for the partial DMR architecture (Chapter 5).

• A new hardware architecture (partial TMR) with a novel scrubbing technique that takes into account area information for a second order polynomial IP. Investigation of the trade-off between availability, area and power consumption of the proposed technique with existing scrubbing techniques (Chapter 5).

1.2 Publications

The following publications have been made during this thesis:

• Michail Vavouras and Christos-Savvas Bouganis, ”Area-Driven Partial Reconfiguration for SEU Mitigation on SRAM-based FPGAs”, In International Conference on Reconfigurable Computing and FPGAs (ReConFig 2016), Cancun, Mexico
1.3 Outline

Background information and the state-of-the-art for fault-tolerance and SEU mitigation on SRAM-based FPGAs is presented in Chapter 2. Chapter 3 introduces the design and implementation of a baseline artificial pancreas application. Moreover, the proposed ASIC/FPGA fault-tolerant artificial pancreas is introduced, which achieves longer lifetime. A novel framework is designed and presented that enables the experimentation with scrubbing strategies on a ZYNQ FPGA in Chapter 4. Two novel scrubbing techniques are presented in Chapter 5 that explore the trade-off between availability, area and power on the framework developed in Chapter 4. Finally, Chapter 6 presents the conclusions drawn from this thesis and key ideas for extending this work.
Chapter 2

Background

2.1 Introduction

In this chapter, a survey of the literature for fault-tolerance and reliability on FPGAs is presented. More specifically, the review includes sources and types of faults in FPGAs and the risks they impose from the application perspective. Furthermore, design techniques targeting detection, masking and correction of faults or errors at the user logic and the configuration memory of FPGAs is presented. FPGA scrubbing is surveyed extensively in terms of existing methods, architectures and SEU mitigation frameworks.

2.2 Faults, Errors and Failures

The terminology adopted in this thesis for describing an abnormal condition for a system comes from the functional safety standard ISO 26262 [11]. Based on the standard, a fault is an “abnormal condition that can cause an element or an item to fail”. The definition for the error is “the discrepancy between a computed, observed or measured value or condition and the true, specified or theoretically correct value or condition”. Finally, the failure is “the termination of the ability of an element to perform a function as required”. The occurrence sequence between the three definitions in a real system is that first a fault occurs. The error is caused due to the fault and the failure is caused by the error. Therefore, a fault will not necessarily result into a failure. For example, a bit-flip in the
configuration memory of an FPGA due to a strike of a particle will only lead to a system failure, only if this bit-flip is used in the FPGA.

2.3 Types of Faults

The faults that occur on FPGAs can be split in two main categories based on their duration in the system: transient and permanent faults. Transient faults are the ones usually caused by radiation in the configuration memory of the FPGA and they are removed with some sort of reconfiguration by loading a gold bitstream on the FPGA. On the other hand, permanent faults are caused due to manufacturing defects, ageing or degradation and exist on the device for its entire life-cycle. These faults are usually modelled as stuck-at-1 or stuck-at-0. If a permanent fault affects part of the device, it can be isolated and a relocation might give the opportunity for mapping the functionality to another region of the chip. The latter implies that there is extra space able to accommodate the application.

2.4 Sources of Faults

Most of the vulnerability factors from the world of ASIC apply to the FPGA circuits, since the same process technology nodes and manufacturing procedures are used.

2.4.1 Ageing and Degradation

Advances in semiconductor technology create a big threat for the reliability of integrated circuits. Apart from transistor ageing and degradation mentioned above, leakage currents, process and temperature variation, wear-out phenomena and manufacturing defects could potentially cause permanent faults leading to a system failure [2].

Negative Bias Temperature Instability (NBTI) applicable to PMOS transistors and Positive Bias Temperature Instability (PBTI) applicable to NMOS transistors are two degradation mechanisms caused by trapped charge in the gate-dielectric channel interface region leading to a rise in the threshold voltage and resulting in less current flowing from the source to the drain of a MOSFET transistor [12].
Another mechanism that causes ageing of transistors is the so called Hot Carrier Injection (HCI). An electron or a hole achieves sufficient energy to overcome the potential barrier of the gate-dielectric interface region and degrades the performance by increasing the threshold voltage and reducing the switching speed of the transistor [13].

Time Dependent Dielectric Breakdown (TDDB) is the phenomenon where a conductive route is formed from the gate of the transistor to the substrate due to long-term electric field in the gate dielectric [14]. The gate leakage current causes an increase in power consumption and a slow switching of the transistor.

Last but not least, electromigration is another cause of ageing of transistors, where metal ions are displaced due to the momentum of electrons moving into the conductor [15], [16]. The effect of electromigration can be short or open circuits leading to hard faults.

2.4.2 Manufacturing Defects

Manufacturing defects are witnessed in electronic circuits due to fabrication impairments manifested as stuck-at-0 or stuck-at-1 hard faults. These defects are static as compared to ageing defects and occur a few hours after the circuit is in operation [17].

Another reason why manufacturing defects occur is due to process variation. Parameters such as temperature, power and timing vary between different regions of an integrated circuit leading to faults that might manifest at the application level.

2.4.3 Radiation

As transistors shrink, the amount of charge required to turn them on or off reduces and this is a major threat for FPGAs, since a strike of a particle transfers energy to the node of the transistor which might be enough to flip the value of the node [18]. This is called a Single Event Upset (SEU) when it occurs in memories and Single Event Transient (SET) when occurs in registers. As compared to the previous sources of faults that belong to permanent faults, this falls into the transient faults category since a refresh/reconfiguration will remove the fault.
Sources of Radiation

A radiation environment consists of high energetic particles such as ions, protons and neutrons. Three main sources of radiation have been pointed out in the literature. The first source of radiation is the Galactic cosmic rays (GCRs) that is a mixture of high energy electrons and protons that enter into the Earth from the outside of our solar system [19]. In addition to that, particles with very high energy such as heavy ions, electrons and protons are generated from the sun’s surface in bursts and last for a few hours. A third category is trapped ions and electrons in the so called Van Allen radiation belts that exist between in 100 Km and 65,000 Km from the Earth’s surface [20].

Apart for the aforementioned primary sources of radiation, radiation is generated due to the collision of energetic particles with electronic materials. For example, in CERN, the Hadron Collider accelerates particles that collide with materials causing phenomena such as SEUs [21].

Radiation Effects

Radiation effects on SRAM-based FPGAs should be mitigated in order to avoid application failure.

Total Ionizing Dose (TID) is the total absorbed energy of the electronic matter due to the interaction with high-energy particles [22]. This effect should be taken into account when commercial off-the-shelf (COTS) SRAM FPGAs are selected to be part of a space mission. However, Xilinx offers space qualified boards such as Virtex-4QV with a tolerance of 300 Krad [23] of TID and Virtex-5QV devices with a guaranteed TID of no more than 1 Mrad [24]. It should be noted that 1 rad = $6.24 \times 10^7 \text{MeV/g}$ according to [25].

Single Event Latchup (SEL) is a type of short circuit that is caused by one or many particles when they strike an SRAM-based FPGA. A voltage spike results in performance degradation, which in some extreme cases might be destructive for the device [22].

Single Event Upset (SEU) is a soft error that flips the value of a memory element. The collision of the particle with the memory element releases energy that might be adequate
to change the state of the memory cell [26]. It is the most common radiation effect in SRAM-FPGAs affecting flip-flops, configuration memory and BRAMs [25].

Single Event Transient (SET) is an instantaneous increase in voltage or current that propagate and manifest to a flip-flop element as an SEU [27].

Single Event Functional interrupt (SEFI) is a disruption of the functionality of the FPGA due to a global reset of the entire device while it is operating or a loss of read and write functionality through the SelectMAP interface or a continuous increase or decrease of the frame address register [28].

**Single Event Effects Rates**

SEE rates have been researched in the field of FPGAs in space missions. According to [28], SEU and SEFI rates have been calculated based on the CREME tool for two orbits, the Low Earth Orbit (LEO) and the Geostationary Earth Orbit (GEO). All memory cells and flip flops were used and the targeted device was a space qualified XQR4VSX55 FPGA. The probability of a SEFI was 1 in 36 years for the LEO and 1 in 103 years for the GEO orbit. The probability of an upset in the flip-flop elements is almost 0.1 upsets per device-day. The upset rate for a BRAM cell is 4.05 upsets per device-day in the LEO orbit and 4.49 upsets in the GEO orbit, while the upset rate of a configuration memory cell is 7.56 upsets per device-day for a LEO orbit and 4.28 for a GEO orbit. Thus, most of the upsets were SEUs as compared to SEFIs. The majority of the SEUs were at the configuration memory of a space qualified XQR4VSX55 FPGA and the BRAMs. Therefore, the primary target for mitigation of SEUs on FPGAs should be the configuration memory and the embedded BRAMs [9].

**2.5 Fault Detection**

A fault-tolerant FPGA system should provide fault detection or masking in the user logic and fault recovery/correction at the configuration layer. The basic idea behind fault detection is redundancy, which can be applied in three different forms. Spatial redundancy
is achieved when two or more elements are placed in parallel computing the same result, whereas information redundancy is when some extra bits of information are added in the data streams for detecting errors. Finally, temporal redundancy exists when a computation is repeated to check if the two calculations are identical. It should be noted that the fault detection techniques surveyed, cover both permanent and transient faults.

2.5.1 Spatial Redundancy

A common form of spatial redundancy is Duplication with Comparison (DWC). Figure 2.1 shows the classic DWC technique where two identical modules are operated in parallel and a comparator (XOR) at the output signals a mismatch in case a fault occurred. This technique is able to detect almost 99% of all failures at the cost of duplicating the area of the circuit [29]. Since the comparator is a single point of failure, [30] proposed a two rail checker circuit that guarantees protection against single faults in the comparator.

DWC can tolerate redundancy at the module level as the first step of a fault-tolerant system with the detection of the error triggering a partial reconfiguration of the FPGA as:

![Figure 2.1: Duplication with comparison](image)

However, one of the limitations of the DWC is that it can’t mask an error as compared to the TMR. The TMR technique uses three identical modules that operate in parallel and a majority voter at the output that is able to mask a single error.

The authors in [34] pointed out that the logic of a circuit can be partitioned into smaller parts and the TMR technique can be applied by inserting intermediate voters.
between the triplicated logic. Using Markov chains it has been shown that the reliability
is increased since the probability of having two replicas of the same TMR scheme in error
is smaller than in the original TMR scheme.

Another work [8] indicates that the number and placements of the voters in a TMR
version of a digital filter can play a significant role in the reliability of the system. It
has been shown that inserting more voters does not mean higher protection from SEUs,
and a moderate usage of voters and partitions leads to a four times improvement of the
sensitivity to routing upset over the standard TMR and a 10% drop in performance.

An alternative to the previous work [8] is to apply TMR at the voter as it consists
a single point of failure, and apply the TMR technique at the netlist level instead of
implementing modular redundancy using the XTMR tool [35], the BYU EDIF tool suite
[36], Precision Hi-Rel by Mentor [37] or Sinplify Premier by Synopsis [38].

The major drawback of the TMR technique for fault masking is the area and power
overheads over the baseline circuit which is at least 200% plus the voter overheads. A
technique named reduced precision redundancy (RPR) as an alternative to TMR was
presented in [41] where two of the three replicas are operated in reduced precision while
the third one computes the result in full precision. For the case study of an FIR filter,
the failure rate was improved almost 200 times while the hardware overhead was only
70% of the unmitigated design. It should be mentioned that the full TMR scheme has
area overheads of 200%, but improves the failure rate 1,200 times as compared to the
unprotected design.

2.5.2 Concurrent Error Detection

Concurrent Error Detection (CED) techniques offer detection of faults by the addition
of extra hardware such as parity bits and cyclic or linear codes in memories or data
flows [43]. Figure 2.2 shows a general architecture of a concurrent error detection scheme
that is based on hardware redundancy and a checker circuit is responsible for signalling
detected faults. CED techniques try to save area as compared to modular redundancy
and offer high reliability. A representative example of CED techniques is the use of Error
Correcting Codes (ECC) and Cyclic Redundancy Codes (CRC) in FPGAs to detect faults in the configuration memory.

![General architecture of a concurrent error detection scheme](image)

Figure 2.2: General architecture of a concurrent error detection scheme [43]

The investigation of different types of Hamming codes such as Hamming codes with distance two and three was performed in state machines with different encoding types, such as one-hot, binary or grey coding in [44], in order to test the robustness against single event upsets using fault injection. The authors found that the Hamming code with distance two offers the best solution in terms of fault-tolerance, size and speed.

Parity bits have been added to the outputs of state machines and the next state logic is stored in memory blocks which are used as lookup tables. Therefore, any error in the memories of the output logic is detected by the parity checker [45].

Algorithmic-based fault tolerance using Error Detection and Correction Codes (EDAC) for a matrix-matrix multiplication in a Xilinx Virtex 5 FPGA was proposed by [46]. A comparison with an unmitigated design using fault injection, demonstrated a 99% reduced design vulnerability with 25% resource overheads.
2.5.3 Off-line Testing

A commonly used technique for testing FPGAs for faults while they are not operating or during scheduled maintenance periods is the Built-In Self Test (BIST). It is a test configuration that includes a test pattern generator, an output analyser and paths under test. It is stored in an external memory and is able to test many types of resources of the device such as LUTs, routing switches, PLLs and clock network. Off-line test methods achieve high fault coverage because they are flexible enough to locate even faulty resources that are not used by the application. The only drawback of BIST schemes is that they cannot detect faults at runtime of the applications, since they are enabled only when the FPGA is offline.

Published work in the field includes a BIST structure that includes self-configurable switch matrices, output analysers and test generators without the need of downloading different bitstreams on the FPGA, thus saving testing time. The authors report that the area overhead of their method is 0.5%, while six alternative switch matrix configurations are used for interconnect testing [47].

A BIST method for testing stuck-at faults and bridging faults in FPGA interconnect with 100% fault coverage and time to diagnose a single fault in the interconnect of 1 second in the XC4020E is proposed by [48]. Stroud et al. [49] proposed a low overhead BIST exploiting the re-programmability of an FPGA achieving maximum fault coverage targeting all programmable logic blocks. Figure 2.3 illustrates a test session where C groups of m-bit test pattern generators (TPGs) are driving C groups of n Block under test (BUT) in parallel. The O groups of n output response analysers (ORAs) are fed with the O outputs of each BUT in order to compare the behaviour of the C groups of BUTs.

Off-line fault detection methods like BISTs are mainly used in FPGAs as a test configuration that is loaded when the FPGA has been powered on (off-line method) to check for faults on logic resources and routing switches [50] and then the original bitstream is downloaded in the FPGA. This process is considered to occur off-line.
2.5.4 On-Line Testing

Roving test methods exploit run time reconfiguration to test a part of the circuit while the other parts remain functional. Figure 2.4 illustrates that the chip is split into equal sized logic blocks having one of them spare for testing purposes and the others performing the functionality of the circuit [51], [52]. These methods were enabled by the partial reconfiguration property of modern FPGAs and provide good fault coverage. The difference with off-line techniques for fault detection where the FPGA is not performing application-related tasks is that the FPGA is operational while a roving test is underway.

An integrated approach for testing both logic cells and interconnects in an on-line fashion using Self Testing AREas (STAR) without interrupting the operation of the application on an ORCA 2CA FPGA is presented in [51]. Their approach allows the use of
partially defective logic with spare resources being close to faulty resources achieving high availability.

An on-line testing strategy for detecting faults in the logic and interconnects of a bus-based FPGA with the only addition of a state machine which is considered to be fault-free is presented in [53], promising high availability for mission-critical applications by achieving small mean time to detect a fault.

2.5.5 Comparison of Fault Detection Techniques

A comparison of the several fault detection methods presented before is illustrated in Table 2.1 regarding different metrics such as speed of detection, area and performance overhead, granularity level and fault coverage.

In terms of the speed of detection, modular redundancy (DWC or TMR) and CED techniques are the fastest ones since the detection of the error from the comparator is usually at the same cycle or one cycle later. TMR is a special case where a single error is masked without any latency happening transparently to the application. Roving test methods offer a medium detection speed (a few seconds), since they leverage partial re-configuration to swap in and out regions for testing. Therefore, the detection speed of the roving methods is dependent on the size of the testing areas. The detection speed of the BIST methods is slow, due to the fact that the FPGA needs to be taken offline.

The largest resource and power overheads appear on the modular redundancy schemes, and more specifically on the TMR, that has more than 200% resource overheads as compared to an unprotected design. DWC has reduced overheads in terms of resources and power since two replicas instead of three are running in parallel. CED techniques save area as compared to modular redundancy techniques, since they add a few redundant bits for error detection as opposed to duplicating or triplicating an entire module. BIST structures have very small resources overhead because the test configurations are stored externally to the FPGA chip (in an external memory) and they are not active during the execution of the application. Roving STAR strategies have medium overheads in power and resource utilisation depending always on the size of the testing regions.
The comparison of the fault detection techniques with respect to the granularity of detection shows that modular redundancy has the coarser granularity and this is restricted by the size of the redundant modules. The finest granularities in terms of resolution of detection is found in off-line and on-line methods with the finest structure being a net of the interconnection topology. CED techniques offer medium size granularity which is smaller than modular redundancy techniques and larger than off-line and on-line test methods.

The best fault coverage is achieved using the off-line and on-line testing structures as they can detect all faults, even those that are not occupied by the application that runs on the FPGA. Modular redundancy offers good fault detection, with all manifested errors being detected while CED techniques have medium fault coverage, since they are not applicable to some applications.

Last but not least, the smallest performance overhead among the four techniques reviewed belongs to modular redundancy, since the latency of the operation at the worst case is increased only by one cycle. CED methods have small performance degradation depending on the complexity of the checker. Furthermore, off-line testing techniques have small performance degradation since the small delay to load the test configuration is happening while the FPGA is not performing critical computations. Finally, the roving STAR technique has large performance overhead due to increased length of paths among the testing regions.

Table 2.1: Comparison of fault detection techniques [54]

<table>
<thead>
<tr>
<th>Technique</th>
<th>Detection latency</th>
<th>Resource overhead</th>
<th>Granularity</th>
<th>Coverage</th>
<th>Performance overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modular Redundancy</td>
<td>fast</td>
<td>very large</td>
<td>coarse</td>
<td>good</td>
<td>very small</td>
</tr>
<tr>
<td>CED</td>
<td>fast</td>
<td>medium</td>
<td>medium</td>
<td>medium</td>
<td>small</td>
</tr>
<tr>
<td>Off-line BIST</td>
<td>slow</td>
<td>very small</td>
<td>fine</td>
<td>very good</td>
<td>small</td>
</tr>
<tr>
<td>Roving STAR</td>
<td>medium</td>
<td>medium</td>
<td>fine</td>
<td>very good</td>
<td>large</td>
</tr>
</tbody>
</table>
2.6 Fault Recovery in FPGAs

FPGA-based fault recovery methods can be separated in two main categories, permanent and transient, based on the type of faults that they can tolerate. Permanent fault recovery methods are further categorised into device-level and configuration-level, based on the level of abstraction at which faults are tolerated [55]. Transient fault recovery includes techniques that remove the accumulation of radiation-induced errors from the configuration memory of the FPGA after the detection of an error.

2.6.1 Permanent Fault Recovery

The first group of methods focuses on yield enhancement and deals with faults at the hardware level of the FPGA. In device-level fault tolerance methods, as soon as the fault has been detected the substitution of faults in logic and/or routing follows using different redundancy schemes.

Device-level Fault Recovery

The Extra Rows method is a redundancy-based fault recovery method introducing extra rows and wiring to the FPGA. In order to support this method, selection logic needs to be added between the row decoder and rows, the lengths of vertical wires are increased by one row and additional sets of vertical wires are added as depicted in Figure 2.5. When a fault is detected, the selection logic can bypass the faulty row and the additional sets of wires are used to maintain the original wiring. No reconfiguration time penalty occurs since the reconfiguration takes place during manufacturing [56].

In the Reconfiguration Network method faulty resources can be bypassed using a secondary interconnect network. This is done in three steps: a) a custom testing process is run on the FPGA to identify faulty resources that will be used for the defect map, b) the defect map is loaded into the configuration memory and is used by an online router to set up the routing by avoiding the faulty parts, and c) once the routing has finished, the actual configuration is loaded on the chip [57].
In the Self-Repairing Architecture [58], run-time fault tolerance is provided and faults are actually tolerated with spare columns. When a fault occurs in a Programmable Logic Block (PLB) the entire column is shifted one position to the right and the routing between the PLBs is adjusted to maintain functionality as shown in Figure 2.6.

The Block-Structured Architecture proposed by [59] is shown in Figure 2.7. Grey colour blocks are configured while blocks with dotted lines inside are unused. The top left
inset figure shows the targetted layout, while the main figure shows the way the layout is mapped given a spare column and two faulty blocks. This method has significant area overheads due to spare PLBs and extra routing for connecting PLBs with each other. Moreover, the reconfiguration overhead is small, since alternate configuration are pre-computed offline.

Another fault tolerance method used for yield enhancement is the Fault-Tolerant Segment method [60] with which one fault can be tolerated on each side of each PLB. Figure 2.8 depicts the extra track of segments to each wiring channel. When a faulty segment is detected, segments are shifted from the faulty segment towards the spare one.

The Fault-Tolerant Grid technique utilises coarser level of fault tolerance without any additional element and time delay. An entire spare routing grid can be used to bypass faulty interconnects as shown in Figure 2.9.

The last method is called SRAM shifting where a shifter circuitry is required to
shift the configuration memory. Two allocation methods have been proposed; the king allocation that sets a spare in the middle of a 3x3 square and the horse allocation which sets the spare in the middle of a cross as shown in Figure 2.10 [61].

An example of an SRAM-shifting fault tolerance method is illustrated in Figure 2.11. An algorithm is placed and routed using the king allocation method. The block at the bottom left side of the system is faulty. The application is shifted in order to bypass the faulty block and is shifted up and right, so that the faulty block is now on a spare block.
Figure 2.11: SRAM shifting example [61]

Configuration-level Fault Recovery

This class attempts to map a circuit to a set of fault-free resources and better decisions might be taken about fault handling. The main drawback of the configuration level fault tolerance is that in many cases an external processor should be used to run time reconfigure the FPGA.

The first method reviewed is the so called pebble shifting [62] that tries to move parts of the systems away from faulty resources using a graph where the PLBs are represented as nodes and the interconnections between PLBs are represented as edges (see Figure 2.12). There is no limit in the number of faults that this method can tolerate as compared to the limited coverage of device level fault tolerant methods. If this method is coupled with a routing fault tolerance method, it can improve significantly the lifetime of an FPGA.

Node covering and cover segments creates chains of PLBs which represent rows in the FPGA and ensures that every PLB has a cover node. In the presence of a fault, nodes are shifted one location at a time towards the chain and the last PLB is reserved as a spare [63]. Local routing is considered as part of the node and is shifted together with the node. To ensure that inter-PLB communication is maintained after shifting, cover segments are added.

Mini-Max grid matching [64] is another method of this class that introduces a minimum cost maximum matching algorithm trying to minimise the maximum distance...
Figure 2.12: Pebble shifting (a) a placed circuit with a faulty resource; (b) the initial graph created from the circuit; and (c) the reconfigured circuit with thick grey lines representing the shifted paths [62]

of any of the blocks that after a fault has to be moved somewhere else. It is similar to pebble shifting but the performance degradation is kept to a minimum.

Tiling [65] is used to tolerate faults that occur on logic and interconnects while partitioning the FPGA into tiles. Each tile contains some part of system's logic, spare logic and interconnects. It is similar to Block Structured technique but it does not make use of redundant blocks. Multiple configurations of each tile are stored in an external memory and faults on the inter-tiled interconnection can be tolerated if additional routing is reserved. The reconfiguration penalty is small since the alternative configurations are pre-compiled and stored in an external memory. An example is shown in Figure 2.13

Cluster-based fault tolerance [66] is a good candidate for cluster-based FPGAs made of basic logic elements (BLEs) and local routing and can tolerate both logic and interconnects faults. It uses several sets of fault-tolerance approaches for both intra-cluster
and global faults on interconnects. Finally, the Column-based approach tolerates faults on both the logic and interconnects of an FPGA. Similar to the Extra Rows method, however, no additional hardware is needed. It relies on pre-compiled alternate configurations and on FPGA columns rather than individual PLBs [67].

Comparison of Permanent Fault Recovery Techniques

According to Table 2.2 the best solution for fault recovery is probably a combination of methods given the application constraints. It should be noted that device level fault tolerance methods do not affect the circuit performance heavily but are not as flexible as their counterparts. The aforementioned table shows the area overhead for both device-level and configuration-level techniques and the reconfiguration complexity for the configuration-level methods only, since the device-level techniques do not include explicit reconfiguration of the FPGA. It should be noted that all groups of PLBs are assumed to be square. $P$ is defined as the number of PLBs in the FPGA, $R$ is the number of faulty routes, $K$ is the average number of sinks per route, $G$ is average number of segments per route, $N$ is the number of PLBs FPGAs have per side, $T$ is the number of PLBs tiles have per side and $A$ is the number of alternate configurations are available per tile.

Several conclusions can be drawn by comparing the device-level and the configuration-level fault-tolerant techniques. The performance impact of fault-tolerance on a circuit is less for the device-level methods than the configuration-level methods. The modifications
of logic and/or routing required by the device-level techniques are less extensive as compared to the configuration-level techniques as most of the device-level methods simply sift rows or columns of cells as compared to configuration-level methods that perform some amount of global re-routing and relocation of logic. Thus, configuration-level schemes have a greater impact on performance than device-level techniques.

Another conclusion is that configuration-level methods are more flexible since they can repair a faulty resource after the FPGA device is manufactured. On the other hand, device-level techniques do not have the same flexibility as they are used during manufacture. Furthermore, configuration-level methodologies are able to tolerate more faults than their counterparts that tolerate only one fault per row or column.

2.6.2 Transient Fault Recovery

Transient fault recovery methods are discussed in this section that focus on time redundancy and checkpoints, and tolerating SEUs in the configuration memory of the FPGA using a well known technique called scrubbing.

A widely used technique for correcting transient faults in FPGAs is the re-execution of the application after the fault has been detected [68] or the rolling back to a known and safe checkpoint [69].
Table 2.2: Comparison of permanent fault recovery techniques

<table>
<thead>
<tr>
<th>Method</th>
<th>Area Overhead</th>
<th>Reconfiguration Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extra rows</td>
<td>Row selector, N PLBs, 1 PLB added to vertical wires</td>
<td>no reconfiguration</td>
</tr>
<tr>
<td>Reconfiguration</td>
<td>Switches and local wiring segments, N PLBs</td>
<td>no reconfiguration</td>
</tr>
<tr>
<td>Self-repairing</td>
<td>N PLBs, configuration memory, duplicated PLBs</td>
<td>no reconfiguration</td>
</tr>
<tr>
<td>Block structured</td>
<td>N PLBs, routing</td>
<td>no reconfiguration</td>
</tr>
<tr>
<td>Segment method</td>
<td>1 additional segment per channel</td>
<td>no reconfiguration</td>
</tr>
<tr>
<td>Grid technique</td>
<td>1 additional grid per channel</td>
<td>no reconfiguration</td>
</tr>
<tr>
<td>SRAM shifting</td>
<td>MUXs and shift regs for each PLB</td>
<td>no reconfiguration</td>
</tr>
<tr>
<td>Pebble shifting</td>
<td>1 PLB</td>
<td>$O(P^{3})$</td>
</tr>
<tr>
<td>Static-node covering</td>
<td>N PLBs</td>
<td>$O(P^{0.5})$</td>
</tr>
<tr>
<td>Dynamic-node covering</td>
<td>1 PLB</td>
<td>$O(P^{1.5} \times \log P)$</td>
</tr>
<tr>
<td>Mini-max grid matching</td>
<td>1 PLB</td>
<td>$O(P^{2.5} \times \log P)$</td>
</tr>
<tr>
<td>Tiling</td>
<td>P/T PLBs</td>
<td>$O(P \times A/T)$</td>
</tr>
<tr>
<td>Cluster-based</td>
<td>1 LUT</td>
<td>$O(R^{2}\log(R) \times K^{2}\log(K) \times G)$</td>
</tr>
<tr>
<td>Column-based</td>
<td>N PLBs</td>
<td>O(1)</td>
</tr>
</tbody>
</table>

**FPGA Scrubbing**

FPGA scrubbing is the process of rewriting the configuration memory of an SRAM FPGA in order to remove faults. Scrubbing architectures, techniques and frameworks are discussed in this section.
Scrubbing Architectures

Fault recovery is achieved on SRAM FPGAs using scrubbing. The SRAM configuration memory is rewritten in order to prevent accumulation of SEUs. Scrubbing architectures can be categorised based on the portion of area that they reconfigure. Device scrubbing architectures involve the reconfiguration of the whole FPGA device with an uncorrupted bitstream. Frame-level scrubbing reconfigures one or multiple frames, which are the smallest addressable regions that can be reconfigured on Xilinx SRAM FPGAs. A frame-level redundancy scrubbing method that minimises energy and area overheads compared to other scrubbing techniques was presented in [70].

A classification of scrubbing architectures can be made based on the location of the scrubbing circuitry as shown in Figure 2.14. An internal scrubber architecture is located with the rest of system on the same FPGA board and is vulnerable to SEUs but constitutes a low-cost solution since a single FPGA is used. On the other hand, an external scrubber requires a second device that is usually radiation hardened and initiates the scrubbing of the other device that includes only the application [72].

![Figure 2.14: Internal vs External scrubber](71)

Scrubbing Techniques

Three main types of scrubbing techniques are found in the literature: blind, readback and on-demand scrubbing. Blind scrubbing is a periodical rewriting of the configuration bitstream based on a user defined scrub rate without having any fault detection awareness.
Blind scrubbing has been evaluated in [73] for different SEU and scrubbing rates for an 128-bit AES application. Jacobs et al. [31] proposed blind scrubbing with a user defined rate to prevent accumulation of bit-flips for FPGAs in space.

Readback scrubbing is performed by reading back the configuration bitstream and checking for corrupted bits by comparing with a golden copy of the bitstream stored in a protected external memory. In particular, [73] compared Cyclically Redundancy Check (CRC)-based scrubbing with Error Correcting Codes (ECC)-based scrubbing and Single Error Correction and Double Error Detection (SECDED)-based scrubbing using the Xilinx SEU controller, combined with variations of TMR schemes for an 128-bits AES algorithm. The results show that a faster response time to errors as compared to blind scrubbing can be achieved. However, readback of the bitstream and scrubbing triggered after every bit-flip, even if it is not a critical one, leads to an increased number of data transfers from the external memory.

On-demand scrubbing is triggered by an error detected by a detection mechanism. An on-demand triggered scrubbing technique was proposed in [32] that offers fast recovery time by exploiting DMR and or TMR at different granularities for fault detection and masking. Straka et al. [33] proposed the use of online checkers or TMR for detecting faults and an on-demand scrubbing methodology orchestrated by a partial reconfiguration controller for reconfiguring the faulty module only. A shifted scrubbing method based on the exploitation of the non-uniform distribution of the critical bits of the configuration memory that reduces the Mean Time to Repair (MTTR) by 30% as compared to standard scrubbing was suggested in [74]. The fault recovery starts by reconfiguring frames that include critical bits and then reconfigures the rest of the frames. A resource efficient mitigation of SEUs on FPGAs used in space applications was introduced in [75]. The radiation level is monitored using internal BRAMs on a Xilinx 5QV device and on-demand scrubbing is triggered by an error detection in a TMR or DMR module.
2.7 SEU Mitigation Frameworks

Recent studies that represent the state of the art on SEU mitigation techniques and frameworks are summarised in Table 2.3. A reconfigurable fault-tolerant framework for space applications where the level of fault tolerance is controlled based on the radiation levels in several orbits was proposed by [31]. The hardware architecture was implemented on a Xilinx Virtex 5 FPGA running at 100 MHz. This work assumes that the inter-arrival time of particles follows a Poisson distribution.

An alternative SEU framework was implemented on a Xilinx ML506 board including a Xilinx Virtex 5 FPGA, where the fault injection process was performed using a custom SEU injection tool through the Joint Test Action Group (JTAG) port [33]. A single FPGA fault injection system implemented on a Virtex 5 FPGA running on 50 MHz and using a custom fault injection tool was proposed in [76].

Brosser et al. [73] designed a framework for mitigating radiation effects running at 50 MHz. The assumed SEU distribution was Gaussian and the fault injection was enabled by the use of the Xilinx SEU controller.

Finally, the authors in [75], introduced a framework for monitoring radiation using internal BRAMs on a Xilinx 5QV FPGA without mentioning the assumed fault injection method and the performance of operation.
Table 2.3: Comparison of SEU mitigation techniques and frameworks

<table>
<thead>
<tr>
<th>Related Work</th>
<th>fault detection</th>
<th>scrubbing technique</th>
<th>seu distribution</th>
<th>fault injection</th>
<th>FPGA device</th>
<th>performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>[32]</td>
<td>DMR, TMR</td>
<td>on-demand</td>
<td>N/A</td>
<td>N/A</td>
<td>VII Pro, V4</td>
<td>N/A</td>
</tr>
<tr>
<td>[31]</td>
<td>DMR, TMR</td>
<td>blind, on-demand</td>
<td>Poison</td>
<td>custom SEU injector</td>
<td>V5</td>
<td>100MHz</td>
</tr>
<tr>
<td>[33]</td>
<td>online checker, TMR</td>
<td>on-demand</td>
<td>N/A</td>
<td>custom SEU injector</td>
<td>V5</td>
<td>N/A</td>
</tr>
<tr>
<td>[76]</td>
<td>DMR, TMR</td>
<td>shifted scrubbing</td>
<td>N/A</td>
<td>custom SEU injector</td>
<td>V5</td>
<td>50MHz</td>
</tr>
<tr>
<td>[73]</td>
<td>TMR</td>
<td>blind, on-demand, CRC, ECC, SECDED</td>
<td>Gaussian</td>
<td>Xilinx SEU controller</td>
<td>V5</td>
<td>50MHz</td>
</tr>
<tr>
<td>[75]</td>
<td>DMR, TMR</td>
<td>on-demand</td>
<td>N/A</td>
<td>N/A</td>
<td>Virtex-5QV</td>
<td>N/A</td>
</tr>
<tr>
<td>[70]</td>
<td>TMR</td>
<td>frame-level scrubbing</td>
<td>N/A</td>
<td>custom SEU injector</td>
<td>V5</td>
<td>50MHz</td>
</tr>
</tbody>
</table>
2.8 Conclusion

This chapter presented the state-of-the-art on fault-tolerance with a particular focus on SEU mitigation for SRAM-FPGAs. Definitions of faults, errors and failures was given based on the ISO 26262. Discussion followed about the types and sources of faults in FPGAs. Fault detection techniques from the literature were presented and compared including modular redundancy, CED, off-line and on-line testing. Fault recovery methods were surveyed, including permanent and transient faults. Finally, the focus turned into the SEU mitigation on SRAM-based FPGAs and scrubbing techniques, architectures and frameworks were discussed and evaluated.

The literature review for fault-tolerance in SRAM-based FPGAs showed that there are multiple studies for fault detection and fault recovery for permanent and transient faults. However, there is a clear need for investigating techniques that maximize the yield and lifetime of applications with less overheads than existing approaches.

The field of SEU mitigation on SRAM-based FPGAs is mature enough with multiple studies using old FPGA devices. A promising idea is to build a framework on a new FPGA device such as the ZYNQ FPGA for investigating existing scrubbing strategies and novel ones that try to reduce resource overheads as compared to DMR and TMR and offer high availability.
Chapter 3

A Hybrid ASIC/FPGA Fault-Tolerant Artificial Pancreas

3.1 Introduction

This chapter introduces a novel fault-tolerant human-implantable artificial pancreas integrated circuit that achieves high dependability. The key feature of the design is a hybrid-substrate of an FPGA-like fabric attached to an ASIC for mapping an artificial pancreas having fault-tolerant features. Transient and permanent fault detection is supported via DMR, while off-line testing is used to identify a permanent fault between the two replicas. Permanent fault correction is achieved by ASIC and FPGA reconfiguration and transient fault correction is accomplished by restarting the application. The evaluation of the proposed artificial pancreas is based on the FMEDA procedure [77] and follows the IEC 61508 functional safety standard [78].

3.2 Artificial Pancreas

A physiologic property that is of great importance for the life of people is glucose concentration in the blood. This is a very interesting research area given the large and increasing number of diabetic patients in the world. High glucose levels known as hyperglycaemia
in the blood, triggers the pancreas for releasing insulin to regulate glucose concentration. The pancreas of diabetic people cannot produce insulin and thus, an artificial pancreas is required, which is effectively a closed-loop control system that samples the glucose concentration in the blood and releases insulin accordingly. Glucose-level-sensing implants constitute only half of this control loop, with the other half using implantable micro-pumps for controlling insulin. Even though an actual, artificial pancreas has not been developed yet, the glucose-sensing implants have constantly increased in numbers and improved over the years [7].

Devices such as the one shown in Figure 3.1 provide a treatment for diabetic patients by having a sensor attached to the human body that continuously monitors the glucose levels. The sensor communicates wirelessly with an external handheld device that is connected to an insulin pump and schedules the insulin injection into the human body. A state-of-the-art artificial pancreas implementation is shown in Figure 3.2. It is a closed-loop insulin delivery system that consists of a glucose sensor which is connected to a controller that triggers the injection of the insulin via the infusion pump.

Figure 3.1: Device for continuous glucose sensing
The proposed artificial pancreas is based on the latest achievements in artificial pancreas research [80], [81], [82] while improving on the state-of-the-art by providing a highly fault-tolerant device, suitable for diabetic people. As shown in Figure 3.3, the current system contains a module that processes data from the glucose sensor and calculates the amount of insulin to be injected, and a module for data logging that includes compression, checksumming and encryption. The encrypted data logging feature is a further innovation in our system and is not considered in the scope of this study.
3.3 Baseline Artificial Pancreas

A baseline hardware implementation of the module that processes sensory data and calculates the insulin injection rate is presented in [80] and adopted in this thesis. It is the core of the artificial pancreas implant and it is implemented in VHDL. The requirements for the artificial pancreas is to produce the required amount of insulin every 60 seconds and since it is going to be an implant it should be power efficient. It is a fixed-point hardware architecture since the requirement is to be a low-power chip. For that reason, it is implemented without having parallelization and pipelining but instead it reuses hardware resources in order to save area.

Figure 3.4 shows the hardware architecture that has been proposed by [80] using the described equations. The main modules are a shift register that stores the last five blood glucose samples, a single precision floating-point adder (ADD), a single-precision floating-point multiplier (MULT), a Finite State Machine (FSM) that controls the inputs of the arithmetic operators, four registers and a single precision floating-point comparator that are not demonstrated in the figure but are included in the FSM module. The actual control of the blood glucose is achieved within this hardware block by reading and storing the last five blood glucose samples and calculating the amount of insulin that should be
produced in order to keep the blood glucose under a certain level. It should be noted that the application needs to have low-power consumption but no tight real time constraints since the insulin dose should be injected to the patient every 60 seconds in the worst case. According to [80], the execution of the algorithm takes at the worst-case scenario 42 clock cycles with

$$\text{Execution Time} = 42 \text{ clock cycles}$$

The input of the Biostator II control algorithm is the glucose value together with some parameters that are defined later on. The output of the algorithm is the insulin dosing rate. A set of mathematical equations that model the functionality of the algorithm is presented below.

The first glucose sample that comes as an input in the system can be represented with $G_0$ and the new incoming sample with $G_5$. Equation 3.1 calculates the approximation of the glucose derivative $m$ based on the previous glucose samples and the new sample $G_5$.

$$m = \frac{2 \cdot G_5 + G_4 - G_2 - 2 \cdot G_1}{10} \quad (3.1)$$

The glucose estimate $G_Y$ is measured in milligrams of sugar per decilitre of blood (mg/dL), one of the most common units of measure in medicine and computed by Equation 3.2 based on the last five glucose samples and the glucose derivative $m$.

$$G_Y = 2 \cdot m + \frac{(G_4 + G_3 + G_2 + G_1 + G_0)}{5} \quad (3.2)$$
The non-linear proportional action $IR_1$ and the derivative action $IR_2$ are calculated by equations 3.3 and 3.4 respectively, where $R$ is the basal infusion rate in milliunits of insulin per minute (mU/min). The "U" is a standard convention used to denote the strength of an insulin solution. $Q$ is the gain for the controller function, $B$ is the glucose set point in (mg/dL) and $KR$ and $KF$ are parameters that are set appropriately in order to avoid hyperglycaemia and hypoglycaemia respectively.

\[
IR_1 = \begin{cases} 
R \cdot \left[ \frac{G_Y - B}{Q} + 1 \right]^2, & \text{if } \frac{G_Y - B}{Q} + 1 \geq 0 \\
0, & \text{otherwise}
\end{cases} \tag{3.3}
\]

\[
IR_2 = \begin{cases} 
R \cdot KR \cdot \frac{m}{1000} \cdot (G_Y - B), & \text{if } m \geq 0 \\
R \cdot KF \cdot \frac{m}{1000} \cdot (G_Y - B), & \text{otherwise}
\end{cases} \tag{3.4}
\]

Equation 3.5 computes the glucose value $G_{30}$, 30 minutes ahead in time based on the glucose derivative $m$ and it is a threshold for the amount of insulin that should be injected. The parameters used are: $B = 80$, $Q = 75$, $R = 4.5$, $KR = 165$, $KF = 45$, $IR_{max} = 400$ and were taken from [80]. The output of the insulin rate dose is calculated by Equation 3.6.

\[
G_{30} = G_Y + m \cdot 30 \tag{3.5}
\]

\[
IR_{calc} = \begin{cases} 
0, & \text{if } G_{30} < 70 \\
\frac{IR_{calc}}{2}, & \text{if } 70 < G_{30} < 90 \\
IR_1, & \text{if } G_Y - B < 0 \\
IR_1 + IR_2, & \text{if } G_Y - B \geq 0 \\
IR_{max}, & \text{if } IR_{calc} > IR_{max}
\end{cases} \tag{3.6}
\]
3.4 Fault Model and Safety Requirements

A fault model needs to be defined for the artificial pancreas. The hardware faults that are considered, are classified into permanent and transient faults based on their effect (duration) on the application. A Single Stuck-at-Fault (SSF) model that includes stuck-at-0 and stuck-at-1 faults is considered for the permanent faults. Several types of physical faults can be represented through the adopted SSF model such as ageing effects or flaws that become apparent on chip during its operation. On the other hand, transient faults are considered to be temporal faults that are referred as soft errors or Single Event Upsets (SEUs). Thus, the SEU fault model is utilized to emulate the effect of transient faults.

The objective is to design an artificial pancreas where single transient faults would be detected and corrected during the entire life of the circuit. In addition, the system should be able to detect and correct a single permanent fault, which leads to safe and correct operation for a long period of time.

Safety related definitions, such as the Safety Integrity Level (SIL), are defined in the functional safety standard 61508 [83] and are adopted by the automotive industry, rail and health-care industries. The SIL level is a measure of the safety risk of a given process that has four levels ranging from 1 to 4, with 1 being the lowest safety integrity level and 4 being the maximum safety that a system can have. SIL level depends on the Safe Failure Fraction (SFF) which is the percentage of possible failures that are safe and have no effect on the system and on the Hardware Fault Tolerance of N (HFT-of-N) value that shows how many faults could cause a loss of the safety function. When the SFF value is greater than 90% and the HFT-of-N is 1, the SIL 3 level is guaranteed [84].

Medical implants follow the IEC 62304 norm for software safety [85]. The maximum level of safety in the IEC 62304 is Class III which is a one to one mapping to SIL 3 level in IEC 61508 standard. For hardware safety, medical devices follow the IEC 60601 norm which does not explicitly state the required safety level [86]. However, prEN 50129 suggests that a similar approach to IEC 62061 [87] should be followed for medical implants. So, based on IEC 62061, SIL 3 represents the integrity level required to avoid serious incidents involving a number of fatalities and/or serious injuries. Therefore, the target system should
operate in a SIL 3 environment with respect to safety.

3.5 Generic description of the fault-tolerant technique

Figure 3.5 shows the block diagram of a modular design that consists of modules $i = 1...n$ with areas $a_1...a_n$. The total area of the design in ASIC is the sum of the individual areas of the modules $\sum_{i=1}^{n} a_i$. The critical path of the design includes all the modules from $a_1$ to $a_n$ and the maximum frequency that the design can operate is $F_{max}$. This baseline implementation is vulnerable to transients and permanent faults since there is no fault detection and will lead to a system failure.

![Block diagram of a modular design](image)

Figure 3.5: Block diagram of a modular design

A fault-tolerant version of the aforementioned design includes duplication at the module-level, some extra area in a reconfigurable fabric such as an FPGA, multiplexers to bypass a faulty module and comparators for detecting faults. Figure 3.6 shows the block diagram of a proposed fault tolerant version of the baseline modular design.
Figure 3.6: Block diagram of a proposed fault tolerant design
The total area of the fault-tolerant design is

\[ \text{Total area} = \text{ASIC area} + \text{FPGA area} \]

It should be noted that the area is measured in number of transistors and the area ratio between FPGA fabric and ASIC is

\[ \frac{\text{FPGA area}}{\text{ASIC area}} = 5 \]

according to [6]. Thus, a module that in ASIC has \( t \) transistors, when placed in the FPGA fabric occupies \( 5t \) transistors. Since, the assumption for the fault-tolerant technique is that one permanent fault should be tolerated, there should space in the FPGA-like substrate for the module with the maximum area. Therefore, the required FPGA-like area is

\[ \text{FPGA area} = 5 \times \max(a_1...a_n) \]

The ASIC area consists of the dual modular redundancy scheme, three two-input multiplexers per pair, one comparator per pair and one n-input multiplexer and one n-output demultiplexer for routing signals between the ASIC and the FPGA part as shown in Equation 3.7.

\[
\text{ASIC area} = 2 \sum_{i=1}^{n} a_i + 3n \times a_{\text{mux}} + n \times a_{\text{comp}} + a_{n_{\text{mux}}} + a_{n_{\text{demux}}} \quad (3.7)
\]

Equation 3.8 defines the total area for the fault-tolerant design, where \( a_i \) is the area of the module \( i \), \( a_{\text{mux}} \) is the area of the two-input multiplexer, \( a_{\text{comp}} \) is the area of the comparator, \( a_{n_{\text{mux}}} \) is the area of the n-input multiplexer between the ASIC and the FPGA and \( a_{n_{\text{demux}}} \) is the area of the n-output demultiplexer that lies between the two substrates.
\[ \text{Total area} = 2 \sum_{i=1}^{n} a_i + 3n \times a_{\text{mux}} + n \times a_{\text{comp}} + a_{n_{\text{mux}}} + a_{n_{\text{demux}}} + 5 \times \max(a_1...a_n) \quad (3.8) \]

Assuming that all modules have the same area \( a_1 = a_n, a_{n_{\text{mux}}} = a_{n_{\text{demux}}}, a_{\text{mux}} < a_1...a_n \) and \( a_{\text{comp}} < a_1...a_n \) which is reasonable in a real-world scenario, a linear increase of \( n \) does not result in a linear increase of the total area of the design. In a typical modular design, the area of a module \( a_i \) and especially its area when this is placed in the FPGA-like fabric will have the greatest contribution to the increase in the total area of the design. The area overheads of the 2-input multiplexers will depend mainly on the size (in number of bits) of the output of the module. The area of the \( n \)-input multiplexer and the \( n \)-output demultiplexer depend on the number of modules of the circuit. The multiplexer’s inputs and the demultiplexer’s outputs apart from the select signals are a power of two, and therefore, an odd number of modules of the design will demand a multiplexer and a demultiplexer with some inputs unused. So, it is more area efficient to split the baseline design in even number of modules, so that all the inputs of the \( n \)-input multiplexer and all the outputs of the \( n \)-input demultiplexer will be used. Furthermore, if the design consists of modules with large area, the overheads introduced by the multiplexers are limited as compared to the overheads introduced by the modules themselves.

When the design is operating in the normal mode without any module placed in the FPGA-like fabric, the critical path will be the same as with the baseline implementation. The two-level scheme with the 2-inputs multiplexers will introduce some delay. The more modules added in series, the more multiplexers will be needed and therefore the larger the delay will be, leading to a reduced operating frequency. In the case where a permanent fault has been detected and the designed has been recovered from that, the critical path will include the routing from/to ASIC to FPGA. In the latter case there will be a drop in the Fmax of the fault-tolerant design. The more complex the datapath in the ASIC, will lead to a more difficult placement and routing of the design and larger performance degradation due to the communication between the ASIC and FPGA parts.
Hardware dependability metrics include diagnostic coverage, safe failure fractions and probability of failures per hour. The diagnostic coverage and the SFF for any design with an arbitrary number of modules that is designed with our fault-tolerant technique is 99% as DMR is in place. The probability of failures per hour depends on the number and the size of the modules. A module with large area will have a higher probability of failure than a smaller sized module. Moreover, if the FPGA-like area is increased so that more permanent faults can be tolerated, the system will have many alternative modes of operations and therefore the probability of a failure will be smaller with an increase of the area due to the requirement of a larger FPGA-like substrate.

Timing dependability include metrics such as fault detection time interval and mean time to repair a fault. The detection latency depends on the detection logic and the latency of the computation of the assumed system. For instance, a fault might not be detected immediately by the DMR per module but at the end of the computation (worst case scenario). Therefore, if the latency of the computation is large, the detection time interval will be large. The time to repair a permanent fault depends on the time to identify a fault and the size of the FPGA-like fabric. The time to identify a fault is proportional to the number of test vectors to pass through the pair of the modules that there was a fault. However, the mean time to repair a fault does not increase linearly with the total size of the design. Mean time to repair includes the reconfiguration of the FPGA-like area and the changing of the select signals of the multiplexer sin the ASIC part with the former taking larger amount of time than the latter to complete. Therefore, assuming that we have a baseline circuit and the fault-tolerant version of it based on our technique. The requirements of the design change and we have to make a change by introducing an extra module in the original design, that is smaller in size than the rest of the modules. We also have to edit the design of the fault-tolerant version by adding DMR and adjusting the multiplexers. However, since the new module is smaller in size than the module with the largest area, the FPGA-like area will be the same and even though the total area of the design has been increased, the mean repair time is not increased linearly.
3.6 Proposed Artificial Pancreas

Biomedical systems such as an artificial pancreas are safety-critical and need continuous and correct operation for long periods of time as this is essential for the patient’s life. In the particular case of implants, the targeted environment imposes constraints so that low-power, low-area and resilience design techniques are usually used. A new architecture is proposed in order to make the baseline artificial pancreas fault-tolerant to transient and permanent faults and SIL 3 compliant. The proposed system architecture is based on a hybrid substrate, where the ASIC part is dedicated to the normal operation of the system, and the FPGA-like fabric can be reconfigured in order to instantiate any possible part of the system, providing further fault-tolerant capabilities to the system.

The proposed artificial pancreas has been designed using the Very High Speed Integrated Circuit Hardware Description Language (VHDL) programming language and has been prototyped on a Xilinx ML605 board that contains a Virtex 6 FPGA. Figure 3.7 illustrates the block diagram of the FPGA-based system prototype. A Microblaze processor is used for testing purposes and is the master on the AXI4-lite bus with all the other blocks attached as slaves on the same bus. The artificial pancreas is shown in the figure as a hybrid-substrate with an FPGA-like fabric attached to an ASIC, and is emulated on the prototype as a partially reconfigurable design with the static part emulating the ASIC and the dynamically reconfigurable region emulating the FPGA-like fabric. In this thesis, the Xilinx Virtex 6 FPGA architecture is considered, but in the future, a more lightweight custom FPGA-like architecture can be used that has less overheads than the Xilinx architecture. The HWICAP module is accessible from the Microblaze to partially reconfigure the artificial pancreas via the ICAP primitive. The test vectors and the partial bitstreams are loaded on the FPGA from an external memory that is protected with Error Correcting Codes (ECC). Finally, an AXI Timer is added for the time measurements and the entire system runs at 20 MHz given its low-power consumption requirements.

The Microblaze resets the proposed artificial pancreas and the starting mode is by operating in a DMR mode by having all the modules in ASIC part and the FPGA-like part empty. The processor sends the input samples via the AXI4-lite bus and the proposed
artificial pancreas IP is calculating the insulin and writes the output into a memory block that is being read by the processor via the same bus. When there is a fault that is detected by the DMR at one of the modules of the artificial pancreas IP the error is being logged and the fault that created the error is considered a transient. Therefore, the Microblaze restarts the application so that the error is removed. If the same error appears again in the same pair of modules, then it is considered to be a permanent, and the identification process starts. The pancreas is taken offline by the processor, a new healthy module identical to the faulty one is loaded on the FPGA-like area and test vectors are passed to identify which of the two ASIC modules is affected by a permanent fault by testing one by one the ASIC modules with the module that was loaded in the FPGA-like area. When the offline testing process finishes by identifying the faulty module, the multiplexers of the ASIC part are configured appropriately to bypass the faulty replica and the IP continues to work free of faults in a DMR mode.

![Figure 3.7: FPGA-based system prototype](image)

3.6.1 The Hybrid-Substrate Fabric

The proposed architecture is illustrated in Figure 3.8. It duplicates each module of the baseline artificial pancreas placed in the ASIC part: two MULTs, two ADDs, two FSMs, two SHIFT_REGs and they operate in a DMR mode able to detect a fault. Apart from the modules themselves, there are multiplexers in place to connect either the two ASIC
modules or the one ASIC module with the module placed on the FPGA-like part. The select signals of the multiplexers are controlled by a configuration register. The multiplexers and the comparators are not protected in the architecture, since they are smaller in size than the main modules of the artificial pancreas. However, the impact on area and failure rate that both the multiplexers and the comparators have is taken into account in the evaluation of the architecture. The error signals are connected to interrupts that are being served by the Microblaze processor.
Figure 3.8: Proposed artificial pancreas
The FPGA-like fabric is designed large enough, such as it can accommodate the multiplier, which is the most resource-hungry module. It is worth noting that only one of the four modules can be placed in the FPGA-like substrate at a time. Allowing multiple modules to be instantiated on the FPGA-like substrate would improve the failure rate of the system but at the same time would cause excessive area and power overheads.

3.6.2 Fault detection

The above architecture supports fault detection for both permanent and transient faults. All output signals of each module of the application are compared with the DMR replica ones. This technique according to [88], gives 99% fault coverage at the expense of duplicating the circuit’s area. The identification of the faulty module being damaged by a permanent fault, is achieved by the off-line testing procedure, as described below.

Special care has been taken for the identification of the faulty module of the arithmetic operators through off-line testing. Figure 3.9 illustrates the off-line testing hardware module for the proposed artificial pancreas. The whole process is executed when the pancreas application has been taken off-line and run in the test mode. Additional multiplexers are added to the arithmetic datapath, that place the arithmetic units under functional or test mode in order to support the off-line testing.
Figure 3.9: Off-line testing of the arithmetic units of the proposed artificial pancreas.
More specifically, during normal operation the arithmetic units, i.e. floating-point adders and multipliers, have their inputs connected to the operand ports and their outputs connected to the output ports. In this case, the verification mechanism is a comparator between the outputs of both arithmetic units which implements DMR. This mechanism is able to identify discrepancies between both units, but it’s unable to identify the faulty unit. In the test mode, the arithmetic units are disconnected from the datapath and connected to a Linear Feedback Shift Register (LFSR), using a seed that has been generated off-line, which generates two pseudo-random test vector sets. The off-line testing circuit accumulates all the results produced by the arithmetic units and compares both results with an expected result which has been computed off-line.

3.6.3 Fault Correction

Fault correction of both transient and permanent faults is supported by the proposed architecture. The level of correction for transient faults is at the system level (whole artificial pancreas) since a restart of the application is required. The permanent faults are corrected at the module level (e.g. adder) through ASIC and FPGA-like reconfiguration.

Permanent fault correction is carried out in two steps and is demonstrated in Figure 3.10. First, FPGA-like reconfiguration is used to instantiate a new identical to the faulty one module in the FPGA-like substrate by using the special hardware module (HWICAP) that is reconfiguring partially the FPGA via the Internal Configuration Access Port (ICAP). Secondly, the static part (ASIC) is reconfigured in the sense that the routing is changed that isolates the faulty ASIC module and connects the new FPGA module. This is done by updating the value of the enable signals of the multiplexers.

Fault correction support is provided only for the main hardware blocks of the artificial pancreas. The multiplexers are not protected and are vulnerable to faults that cannot be detected and corrected. However, the area of a multiplexer as compared to the main blocks of the application is very small and therefore the probability of failure due to a fault at the multiplexers is small.
3.7 Evaluation Results

The proposed medical implant was evaluated under three sets of metrics: a) hardware dependability, b) timing dependability, and c) application-related metrics. The hardware dependability metrics are the Diagnostic Coverage (DC) which is the percentage of faults that have been detected by the detection mechanism for a particular module of the application or the whole application, the SFF that is defined as the total ratio of safe failures over the total failures, and the PFH which is the probability of a failure of the application per hour.

The second set of metrics consists of the Process Safety Time (PST) which is the time budget in which the fault-tolerance functions of the proposed system should be completed before a failure will be catastrophic for the system. PST should be greater than the Worst Case Execution Time (WCET) of the application, which is the maximum value of the sum of the Detection Time Interval (DTI) plus the Mean Repair Time (MRT). DTI is measured as the number of clock cycles elapsed from the occurrence of a fault until the detection of the fault while MRT is measured as the number of clock cycles needed to repair the system. The third set of metrics includes area measurements.

When considering permanent faults, the proposed architecture is compared to a DMR version of the baseline pancreas at the system level. For transient faults, a comparison of the suggested artificial pancreas is made with a TMR version of the baseline at the system level.
level. Hardware and timing dependability metrics have been calculated separately for the permanent and the transient faults.

### 3.7.1 Target technology node

It is worth mentioning that a constant elementary failure rate per transistor or per flip-flop is assumed at 18 nm process technology based on [1], [11] and [77]. When considering permanent faults, the failure rate of a transistor is \( \lambda_{\text{perm,el}} = 1.91 \times 10^{-6} \) FIT. For transient faults, the failure rate of a flip-flop is \( \lambda_{\text{set,el}} = 3.05 \times 10^{-5} \) FIT and the failure rate of a transistor is \( \lambda_{\text{set,el}} = 3.05 \times 10^{-5} \) according to [1], [11] and [77], where 1 FIT is one failure in \( 10^9 \) hours.

### 3.7.2 Permanent Fault Evaluation

**Hardware Dependability metrics**

The diagnostic coverage (DC) for the permanent faults is 99% for all the modules of the application [29], [88]. SFF is 99%, since all the modules have the same diagnostic coverage and it is assumed that every fault that occurred in the system is dangerous. The PFH is calculated as the probability of not detecting a permanent fault plus the probability of detecting a permanent fault but not being able to correct it. The probability of failure per hour caused by a permanent fault was measured to be \( 4.96 \times 10^{-14} \) considering all possible configurations of the system for each one of the four ASIC modules being placed in the FPGA-like substrate.

**Timing dependability metrics**

The execution time of the application is 42 clock cycles. Therefore, in the worst case scenario a fault is detected at the end of the computation and DTI is 43 clock cycles (one clock cycle is needed for reading the comparator output).

However, the time spent for identifying the faulty module is calculated by the off-line test which is based on fault injection at the Register Transfer Level (RTL) level using
Modelsim simulation. The type of faults that were injected are stuck-at-1 and stuck-at-0 faults. The size of test vectors used was varied from 4,096 to 131,070. The identification coverage, that is the probability to identify which of the two modules is faulty, was found to be 88% and was achieved by using 65,535 test vectors, sampled from a pseudo-random uniform distribution, since further increase of the number of test vectors did not lead to higher coverage. The whole process takes 65,535 clock cycles due to the size of the test vectors that are used.

MRT is the reconfiguration time that ranges from 192 ms when FPGA-like reconfiguration is concerned to 1.3 µsec when ASIC reconfiguration is performed. It needs to be noted that the size of a partial bitstream for the FPGA-like substrate is 345 Kbytes. Thus, the WCET for this case is 19,200,043 clock cycles.

For the pancreas application, each application-iteration has a real-time deadline of 60 seconds to complete. Based on the 20 MHz clock frequency, the safety mechanisms need to check and repair the components in less than $1.2 \times 10^9$ clock cycles (PST). The WCET should be less than the PST in order to meet the timing dependability requirement of the application, and this condition is satisfied for the case of the permanent faults.

**Application-related metrics**

Area metrics were obtained targeting the ST CMOS 65 nm process technology because this was the most modern standard cell library that was available in the lab. The area of the proposed artificial pancreas is measured in number of transistors, that does not depend on the process technology. The total number of transistors of the suggested architecture is 236,819 as depicted in Table 3.1. Half of the total area is required by the ASIC modules and the other half by the FPGA-like substrate. The ratio of the area of an FPGA-like module to its ASIC counterpart for the same process node is assumed to be 5 [6]. Table 3.1 summarizes the breakdown of the ASIC area of the proposed application in terms of number of transistors and number of flip-flops.

A DMR version of the baseline pancreas is considered when comparing against the proposed architecture for permanent faults. The DMR pancreas has 99% DC and SFF
Table 3.1: Breakdown of resources of the ASIC part of the proposed architecture

<table>
<thead>
<tr>
<th>Sub-part</th>
<th># of transistors</th>
<th># of flip-flops</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplier</td>
<td>52,638</td>
<td>68</td>
</tr>
<tr>
<td>Adder</td>
<td>35,491</td>
<td>68</td>
</tr>
<tr>
<td>FSM</td>
<td>16,141</td>
<td>852</td>
</tr>
<tr>
<td>Shift register</td>
<td>14,318</td>
<td>754</td>
</tr>
<tr>
<td>Total</td>
<td>118,588</td>
<td>1,742</td>
</tr>
</tbody>
</table>

while its PFH is $2.53 \times 10^{-10}$. It needs to be noted here that a DMR system can only detect a fault with a 99% probability and it cannot identify which of the two modules is faulty and therefore, cannot correct a fault. Hence, the probability to correct a fault is zero. Thus, our approach has 5,100x lower failure rate per hour than the DMR pancreas mainly because the DMR can never correct a permanent fault. The area of the DMR pancreas is 97,692 transistors. In comparison, the proposed artificial pancreas has 2.4x the area of the DMR version. Table 3.2 shows the comparison of our approach for the permanent faults with the DMR version of the pancreas application.

Table 3.2: Hardware and Timing Dependability metrics for permanent faults of the proposed and DMR versions of the pancreas application

<table>
<thead>
<tr>
<th></th>
<th>Proposed</th>
<th>DMR</th>
</tr>
</thead>
<tbody>
<tr>
<td>DC</td>
<td>99%</td>
<td>99%</td>
</tr>
<tr>
<td>SFF</td>
<td>99%</td>
<td>99%</td>
</tr>
<tr>
<td>PFH ($h^{-1}$)</td>
<td>$4.96 \times 10^{-14}$</td>
<td>$2.53 \times 10^{-10}$</td>
</tr>
<tr>
<td>DTI (clock cycles)</td>
<td>43</td>
<td>43</td>
</tr>
<tr>
<td>MRT (clock cycles)</td>
<td>19,200,000</td>
<td>N/A</td>
</tr>
<tr>
<td>Area (# of transistors)</td>
<td>236,819</td>
<td>97,692</td>
</tr>
</tbody>
</table>

3.7.3 Transient Fault Evaluation

Hardware Dependability metrics

The DC for the transient faults is assumed to be 99% for every module of the application [29], [88]. For the same reason with the permanent faults case, the SFF for the transients
is 99%. The PFH for the transient faults is $2.27 \times 10^{-12}$ and has been calculated for the possible modes of operation of the proposed architecture.

**Timing Dependability metrics**

DTI is 43 clock cycles in the worst case scenario. In this case, there is no time spent identifying the faulty module since the repair by reset of the application will eliminate the fault if that was a transient. MRT is the time the soft processor needs to reset the application in order to correct the transient which is 130 clock cycles. The WCET is 173 clock cycles and therefore the PST of the applications is satisfied.

**Application-related metrics**

The area of the TMR is 146,538 transistors, meaning that the proposed architecture has 1.6x the area of a TMR. A comparison should be made with a TMR version of the application to highlight the benefits of our techniques for the transient faults. The DC and SFF of a TMR system is 99% and the probability to correct a fault after it has been detected is 99%. The PFH is $1.88 \times 10^{-10}$ and thus, the proposed architecture has a lower failure rate per hour of 83x as compared to the TMR. Table 3.3 summarizes the comparison of our architecture with respect to transient faults against the TMR version of the pancreas application.

Table 3.3: Hardware and Timing Dependability metrics for transient faults of the proposed and TMR versions of the pancreas application

<table>
<thead>
<tr>
<th></th>
<th>Proposed</th>
<th>TMR</th>
</tr>
</thead>
<tbody>
<tr>
<td>DC</td>
<td>99%</td>
<td>99%</td>
</tr>
<tr>
<td>SFF</td>
<td>99%</td>
<td>99%</td>
</tr>
<tr>
<td>PFH ($h^{-1}$)</td>
<td>$2.27 \times 10^{-12}$</td>
<td>$1.88 \times 10^{-10}$</td>
</tr>
<tr>
<td>DTI (clock cycles)</td>
<td>43</td>
<td>43</td>
</tr>
<tr>
<td>MRT (clock cycles)</td>
<td>130</td>
<td>N/A</td>
</tr>
<tr>
<td>Area (# of transistors)</td>
<td>236,819</td>
<td>146,538</td>
</tr>
</tbody>
</table>
3.8 Conclusion

In conclusion, a hybrid-substrate has been proposed for providing fault-tolerance capabilities to a safe-critical application, namely an artificial pancreas design. The proposed architecture is able to detect and correct transient and permanent faults, utilizing DMR on a per-module basis and ASIC and FPGA-like reconfiguration to correct permanent faults. The implementation results show that in terms of hardware dependability a total safe failure fraction $SFF_{total}$ of 99% and total probability of failure per hour $PFH_{total}$ that equals $2.32\times10^{-12}$ can be achieved. The value of $SFF_{total}$ means that the required SIL 3 level is achieved for the particular application. In comparison to existing fault-tolerance techniques, the proposed architecture achieves 5,100x lower PFH than a DMR for permanent faults with 2.4x the area of the DMR. In addition, the proposed solution achieves 83x lower PFH than a TMR with 1.6x area overheads when considering transient faults. The reported overheads in area is due to the FPGA-like fabric that is being used in the proposed architecture. The current area ratio between FPGA-like substrate and ASIC is 5 and if this ratio decreases in the future the area overheads of the proposed architecture will be reduced. Thus, the above case study showed that a flexible reconfigurable fabric next to an ASIC offers flexibility that can potentially harness the fault-tolerance of a digital system.

Motivated by the study of the permanent and transient faults for the artificial pancreas IP, we moved on by introducing an FPGA framework able to perform fault injection at the configuration layer of an SRAM FPGA in order to emulate the effects of SEUs in digital systems. The details of the framework is presented in the next chapter.
Chapter 4

FPGA Platform for Investigating the Impact of SEUs to a Design

4.1 Introduction

This chapter presents an FPGA platform enabling the investigation of the impact of SEUs to a design. The motivation behind the framework came from the work presented in the previous chapter, for which a real platform for emulating SEUs and exploring the impact of various SEU mitigation strategies applied to a specific design was needed. The implementation was done on a Xilinx Zedboard FPGA board that includes the ZYNQ architecture.

Most of the electronic systems that operate in a radiation environment need to have some protection against SEUs. Space companies use radiation hardened devices to map their systems or redundancy at the device level. A typical fault tolerant system consists of two FPGA boards with one managing the I/O and the control logic and the other having the actual circuit under test. The former is usually protected with TMR, while the latter has SEU mitigation strategies and algorithms either at the device or the application level.
4.2 High level System Overview

The proposed framework emulates SEUs caused by radiation particles striking the resources of an SRAM FPGA not only for space applications but also for laboratories, such as CERN. The part of the device that is more vulnerable to this type of faults is the configuration memory of an SRAM FPGA as compared to flip-flops and BRAMs [9]. The main features of the framework are the fault injection at the SRAM configuration memory and FPGA scrubbing at various rates. Fault injection is supported using the Xilinx Soft Error Mitigation core where single bit-flips can be injected at the SRAM cells of the FPGA and more particularly at specified locations based on the partial reconfiguration flow. FPGA scrubbing is used through the dynamic partial reconfiguration property of Xilinx FPGAs and more specifically using the ICAP port to rewrite the configuration memory and correct the errors at the application level that were caused by the SEUs. The framework supports several SEU rates and scrubbing rates in order to emulate several real radiation environments. The fault injection platform can be either used in a static way for sensitivity analysis or in a more dynamic fashion for implementing different scrubbing strategies and evaluating dependability metrics such as availability. The SEU mitigation framework consists of hardware and software modules that will be discussed in more detail in the following subsections.

4.2.1 Xilinx Zedboard

Figure 4.1 illustrates the Zynq Evaluation and Development FPGA board. It is based on the Xilinx ZYNQ 7000 All programmable SoC, combining a processing system with a dual core ARM Corex-A9 and a programmable logic with 85,000 logic cells. It has 512 MB of high speed DDR3 external memory and auxiliary SD card. It has an Ethernet interface supporting 10/100/1Gbps and USB and JTAG connections for peripherals and programming. The PL is clocked through an on board oscillator of 100MHz and it supports human computer interaction interfaces such as 7 push buttons and 9 LEDs.
4.2.2 Xilinx ZYNQ FPGA Architecture

The Xilinx ZYNQ 7000 All programmable SoC is split into two parts. The Processing System (PS) with an ARM cortex A9 dual processor running at 667 MHZ. It has a 32KB Level 1 4-way set associative instruction and data caches (one for each core) and 512 KB 8-way set associative Level 2 cache which is shared between the two cores. Apart from these memories, the hierarchy has an extra 256 KB on-chip memory. The Programmable Logic (PL) consists of 85K Configurable Logic Blocks (CLBs) and more specifically, 53,200 Look-up tables (LUTs) and 106,400 Flip-Flops (FFs) together with carry chain macros and multiplexers. The whole device has 4.9 Mb of on-chip memory that can be used as 36Kb Block RAM or can be configured as dual 18 Kb block RAM. In addition, the architecture has 220 DSP blocks able to perform any multiply accumulate operation.

4.2.3 Hardware

The experimental system consists of a desktop PC and a Zedboard FPGA board. The PC is responsible for configuring the parameters of the experiment, such as SEU events and CUT, as well as for retrieving the data from the board and analysing them. In addition to that, automation is achieved using tcl scripts that manipulate any type of experiments that the user wants to run on the FPGA. The FPGA board has the supporting framework,
that is the part responsible to inject the SEUs to the configuration memory allocated to the CUT, performs the desired SEU mitigation strategy and logs any activity.

Figure 4.2: Hardware prototype

Figure 4.2 illustrates the block diagram of the hardware architecture implemented on the Zedboard. Due to the System on Chip (SoC) availability of the Zedboard, the envisioned system with the two FPGAs can be mapped on a single board with a SoC device. The host is a desktop personal computer with an Intel i7 2600 running Matlab. The ARM processor runs the LWIP stack for communicating with the host PC via Ethernet over the UDP/IP protocol [89]. The SEM core is used for injecting the configuration bit-flips using the ICAP port. The ICAP port is shared between the SEM core and the HWICAP module. The Hardware Internal Configuration Access Port (HWICAP) controller performs the scrubbing by transferring the bitstream from the DDR3 to the ICAP port. Three 32-bit AXI timer modules are connected on the AXI4lite bus and five interrupts from the programmable logic (PL) to the processing system (PS) are used for keeping timestamps (time of events) for the experiments. The User IP includes an input BRAM that stores and feeds input data to the CUT and the gold designs. Moreover, the CUT and Gold BRAMs are for debugging purposes and a comparator checks for discrepancy between the outputs of the CUT and the gold circuits.

4.2.4 Configuration Memory

The configuration memory of the ZYNQ FPGA is the set of the SRAM cells that configure the FPGA resources such as LUTs, switch matrices for routing, hard macro multipliers and flip flops in order to implement the user defined algorithm. It is organised in frames, that
is the smallest addressable region that can be reconfigured and each frame contains 101 words of 32 bits each. Therefore, each frame has 3,232 bits. There are frames that include configuration memory bits but also frames containing bits that control the DSP and Block RAM memory resources. As mentioned in Section 4.1, only the bits that correspond to the LUTs and routing are considered as candidate fault locations since more than 90% of SRAM cells belong to routing [8], [90].

**Xilinx essential bits**

It should be noted that Xilinx offers the essential bit technology that clusters the configuration SRAM bits into essential and not essential for an application. An essential bit, is a bit that if flipped might cause a functional error because it actually changes the design that is mapped on the FPGA. Figure 4.3 shows the Venn diagram that describes the relationships between device configuration bits, essential bits, prioritised essential bits and critical bits. A critical bit, is an essential bit that if flipped, produces a functional error visible to an output of the application. Not all essential bits are critical bits and thus, the critical bits are a subset of the essential bits. Prioritised essential bits are a subset of the essential bits of a design that refer to a particular module or instance of the design. Those bits correspond to the nets and the components of a module resulting in a faster fault injection campaign.

Xilinx Vivado design suite provides a way of determining which bits of the user’s function (RTL) are essential, thanks to the Xilinx Essential Bits technology. The user describes the functionality of a circuit in RTL, and then the Vivado tool synthesizes this RTL code to a gate level netlist, which in turn is going to be placed and routed on the FPGA. Finally, the tool provides a bitfile that includes all the configuration information for the respective FPGA. Together, with the bitfile, a file with an .ebd name can be generated if the essential bits flag is selected in the bitstream options. The .ebd file includes all the frames that correspond to the resources of the FPGA apart from DSP and BRAM frames. Therefore, the size of the .ebd file is smaller than the bitstream file.

In this way, the essential bits of the design are included in the .ebd file organised in
32-bit words. The bits with the value of '1' in these words indicate essential bits and bits with '0' non essential bits. Based on Xilinx proprietary information, data is listed in the .ebd file as 32-bit words per line. In ZYNQ architectures, there are 101 words (lines) per frame. There is always one entire frame of padding at the start, before frame zero, and this padding frame should be ignored. In each word, the MSB is on the left and the LSB is on the right. Based on this information, a linear frame address is generated that is used from the SEM core in order to inject a particular bit flip in a specific location. However, there is no information provided for which bit corresponds to which particular resource in the FPGA. This requires a reverse engineering process and is out of the scope of this thesis.

Apart from the .ebd file for the whole FPGA, Xilinx offers the possibility to generate partial .ebd files for partial reconfigurable partition. For example, if a design is partitioned into PRs, partial .ebd files can be generated that are smaller in size than the .ebd file that corresponds to the whole FPGA. In that way, less time is required to perform fault injections and the user can flip bits that correspond to one part of the design only, if that
is desirable.

To find the critical bits of an FPGA-mapped design, one should perform fault injection at the configuration SRAM layer of the FPGA and mark as critical the bits that produce an error at the output of the CUT. In this work, the list of the essential bits is sampled to flip configuration bits that correspond to the CUT and reduces the time of the fault injection process as the bit-flips target the CUT.

4.2.5 High level view of fault injection and scrubbing

As can be seen from Figure 4.2, the fault injection process and the FPGA scrubbing process share the ICAP primitive through a MUX structure. The indented functionality is to perform fault injection using the SEM Core and FPGA scrubbing using the HWICAP module. The low level details of the Xilinx SEM core, the HWICAP module and the ICAP primitive is presented in the following sections.

Xilinx SEM Core

The Soft Error Mitigation Core (SEM) was introduced by Xilinx in order to perform SEU injection, detection, correction and classification of SEUs on the configuration memory of Xilinx FPGAs [92]. The motivation behind the core was to increase system availability and provide an alternative to beam testing which is costly and time consuming process. Figure 4.4 illustrates a system level design example block diagram that consists of the SEM Core that its code is encrypted and how it is connected with the other modules of the design example.

Fault injection at the SRAM configuration layer is supported through the ICAP primitive. The control logic of the SEM core, reads a frame from the configuration memory and inverts the value of a particular bit and then it writes it back to the configuration memory using the same interface.

Apart from configuration frames, there are ECC and CRC bits in each frame for detection purposes. Moreover, correction can be done either by reading back the frame in which a bit was injected and flip back its value, or remove ”by replace” by copying a
fresh clean version of the bitstream stored in an external memory. The SEM core has an interface to an external memory that can store the golden bitstream and fetch it when it is needed.

The SEM IP can be controlled through a monitor interface based on UART or via Chipscope. In this work we have edited the SEM IP core and introduced an AXI4-lite wrapper so that it can be connected on an AXI4-lite interconnect and can be controlled by the ARM processor. This provides flexibility, portability and ease of use since the control of the core is now orchestrated by the CPU.

The core of the SEM IP is a state machine that controls the functionality of the IP. The controller starts by initializing itself to the idle state. Then from the idle state it can move to the observation state where it spends most of the time trying to detect faults by scanning the configuration memory and checking the ECC and CRC for errors in frames. From the idle state the controller can jump to the injection state where a bit flip is introduced somewhere as dictated by the linear frame address or to the correction state where the flip-back process takes place. The above switching between states is controlled by the ARM core by writing to memory mapped registers that are connected to a strobe signal and the injection address (linear frame address) of the SEM core interface.
This module includes both hardware and software parts. Figure 4.5 presents the block diagram of the HWICAP module [93]. The main blocks are the ICAP primitive, asynchronous read and write FIFOs, control and status registers and a finite state machine that is the control of the IP, by reading and writing from/to the ICAP. A new feature of the AXI HWICAP v3.0 module provides the arbitration of the ICAP interface for enabling the sharing of the ICAP with other hardware modules. In our case, the HWICAP module shares the ICAP with the SEM core. When fault injection is scheduled, the ARM CPU sets the appropriate signals that are memory mapped to the AXI4-lite interconnect so that the SEM core takes access of the ICAP. To be more specific, the icap grant signal is set together with the icap request signal. On the other hand, when a scrub operation takes place, the HWICAP module grants access to the ICAP in order to partially reconfigure the FPGA. It should be noted that, this configuration has the limitation that fault injection and FPGA scrubbing are two mutually exclusive operations.
ICAP

The Internal Configuration Access Port is an FPGA primitive for dynamically reconfiguring the entire FPGA or part of it, by writing the bitstream from an external memory to the ICAP. The Xilinx ZYNQ FPGA has two primitive ICAPs but only one can be used at a time. When one hardware or software module has control over one ICAP port, the other ICAP port cannot be used by other hardware blocks. It is a 32 bits serial interface, with an input and an output port. It takes a clock input from the clock tree of the device and this clock can be asynchronous to the main clock of the AXI4lite interconnect. In this work, this clock input is connected with the AXI interconnect port for running the entire framework at the same speed of 50 MHz and avoiding clock-crossing domain issues and multiple clock design for simplicity. The interface has also a busy output signal for monitoring when the port is busy with operations. The other signals are the read/write input signal and the chip enable input signal. The former is used for specifying the read or write operation and the latter for enabling the ICAP primitive. The throughput of writing or reading data to/from the ICAP is 1.6 Gbps using a 50 MHz clock.

4.2.6 AXI Timers

The AXI Timer/Counter IP is a 32 or 64 counter module that has an AXI4lite interface and can be directly connected to the AXI4lite bus and therefore communicate with the ARM processor. It can be configured as 32 or 64 bits counter and it actually includes 2 counters that can be programmed to count up or down. Moreover, the option to generate interrupts at specified intervals is provided for the IP. In addition, the counters can be frozen and released based on the needs of the application.

4.2.7 Hardware Support for the interrupts

The ARM processor supports interrupts from different sources and uses the General Interrupt Controller (GIC) in order to serve them. Three types of interrupts are supported: software generated, shared and private peripheral interrupts. The first category has 16 interrupts per processor and they can stop the operation of one or two cores. The shared
peripheral interrupts are 60 in total and are shared between the two cores. They can come from the peripherals or the PL. The third category consists of five private interrupts that are private to each core, such as watchdog timer and CPU timer. In this work, five shared peripheral interrupts are used and connected to one of the two ARM cores. Three of the interrupts come from the three AXI timers and the other two are used for the CUT for logging error events as will be shown in Chapter 5.

4.2.8 Modifications at the hardware level

The modifications made at the hardware level are depicted in Figure 4.6, where the grey shaded blocks are added in order to support the functionality for sharing the ICAP between the SEM Core and the HWICAP module. The major modification was the design of an AXI wrapper, for the design example of the SEM Core, in order to be connected easily to the SoC. Control logic was added inside the AXI wrapper of the example design of the SEM Core for controlling the states from which the controller passes. Moreover, an extra multiplexer was added to support the functionality for switching the access between the SEM controller and the HWICAP module. Furthermore, a state machine was designed for handling the icap grant (icap_gnt), icap request (icap_req) and icap release (icap_rel) signals that are at the interface of the HWICAP module. When the SEM core has the control of the ICAP, the multiplexer has the zero value and the icap_gnt signal is zero. On the other hand, when the HWICAP module has the access to the ICAP the multiplexer is set to '1'. It should be mentioned that ICAP primitive was removed from the HWICAP module, where it is instantiated by default, and the arbiter interface of the HWICAP gives the opportunity to access the ICAP primitive inside the SEM IP core.

Figure 4.7 illustrates the state machine that controls the fault injection process for the SEM Core. It consists of three states: the initialization, the idle and the injection states. The controller enters the initialization state after a global reset where it initialises itself. In order for a fault to be injected, the SEM Core should be moved first to the idle state. In the idle state the SEM core, waits to receive a fault injection command and an address from the CPU and then it jumps to the injection state, where the actual injection
Figure 4.6: Arbitration of the ICAP primitive between the SEM Core and the HWICAP process takes place. When the injection finishes the controller moves back to the idle waiting for a new injection command to come.
Figure 4.7: Fault injection process using the SEM Core

Figure 4.8 demonstrates the state diagram for sharing the ICAP primitive between the SEM and the HWICAP cores. It should be noted that there is a signal called reset fsm that is memory mapped for resetting this logic. When this signal is asserted by the processor, the state machine enters the idle state where the grant and release signals are set to zero. The transition from the idle state to either the sem or the hwicap states depends on the value of the signal that controls the multiplexer. If the icap_mux_sel is zero the machine jumps to the the sem state, else to the hwicap state and remain to these states if the icap_mux_sel signal has a stable value. In both states the grant and release signals that correspond to the HWICAP module are set to zero. The state machine changes from hwicap to sem state when the icap_mux_sel is zero and from sem to hwicap state when icap_mux_sel is one. When the machine is in hwicap state and a write request is asserted from the HWICAP module the machine moves to the write state and the grant signal
becomes one. It remains in this state while the request is high, otherwise it changes to the idle state.

![Control Logic Diagram](image)

Figure 4.8: Control logic for sharing the ICAP between the SEM core and the HWICAP module

4.2.9 Software

Figure 4.9 illustrates the flowchart of the software routine running on the ARM core in bare-metal. The decision for using bare-metal C and not any real-time operating system such as POSIX threads was taken because a fast response to the interrupts was required in order to avoid time-related overheads. The key parts of the software routine are: a) the communication with the host, b) the control of the interrupts and c) the management of the partial reconfiguration process for both injection and correction of faults. More
specifically, the host PC, transfers the input data for the user IP, the bit-flip addresses, the time intervals of the SEU events, the scrub rate, the duration of the experiment (value for the global counter) and the partial bitstream/bitstreams to the DDR3 memory.

Figure 4.9: Flowchart of the software

The input data are sampled randomly from a uniform distribution in MATLAB and are written from the ARM core through the AXI4lite to the input BRAM in the PL. The processor setups the interrupts and the timers of the system and it starts by giving access to the SEM Core over the ICAP. It should be noted that the PCAP is another port for performing partial reconfiguration directly from the ARM processor without the HWICAP module in place. The specification of the Xilinx ZYNQ FPGA dictates that either the PCAP or the ICAP can be used to partially reconfigure the FPGA at a given time. If the HWICAP module is in use the PCAP should be disabled and when there is no HWICAP module, partial reconfiguration can be done via the PCAP interface. The first attempt was to use the PCAP interface together with the SEM core but, this design choice was not supported by Xilinx at that time, so the implementation with the HWICAP was selected.
Table 4.1 gives information about all the interrupts on the system. Every interrupt has its own Interrupt Service Routine (ISR). The highest priority interrupt is the one connected to the AXI timer 0 and the lowest priority interrupt is the one coming from the CUT, indicating an error at the output port. The priority of the interrupts indicates that when two or more interrupts happen at the same time, the one with the highest priority will be served first. One key implementation is that when the ARM core receives an interrupt from one of the modules in the PL, it stops the execution the main function and the code that is written in the body of the corresponding ISR is executed. The first thing to happen inside the core of every ISR, is to mask all interrupts, and before exiting the ISR function re-enabling all of them.

Table 4.1: Interrupts description for the SEU mitigation framework

<table>
<thead>
<tr>
<th>Description</th>
<th>Priority</th>
<th>Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>AXI Timer 0</td>
<td>Measure time elapsed</td>
<td>Highest (1)</td>
</tr>
<tr>
<td>AXI Timer 1</td>
<td>Performs fault injection</td>
<td>Second Highest (2)</td>
</tr>
<tr>
<td></td>
<td>at specified time intervals</td>
<td></td>
</tr>
<tr>
<td>AXI Timer 2</td>
<td>Performs scrubbing</td>
<td>Third Highest (3)</td>
</tr>
<tr>
<td></td>
<td>at specified time intervals</td>
<td></td>
</tr>
<tr>
<td>Oracle Interrupt</td>
<td>Indicates the real error</td>
<td>Fourth Highest (4)</td>
</tr>
<tr>
<td></td>
<td>based on the Gold design</td>
<td></td>
</tr>
<tr>
<td>DMR Interrput</td>
<td>Indicates the DMR error</td>
<td>Fifth Highest (5)</td>
</tr>
<tr>
<td></td>
<td>from the CUT</td>
<td></td>
</tr>
</tbody>
</table>

The three timers are loaded with values that had been calculated offline in MATLAB. For instance, the AXI timer 0 that is responsible for setting the duration of the experiment that the user wants to run, is loaded with the number of times that its internal counter will expire.

Following the same process, the values for the AXI Timers 1 and 2 are loaded appropriately. AXI Timer 1 is responsible for injecting the bit-flips based on the inter-arrival time of the events that follows any distribution. When this counter expires, it generates an interrupt and the next piece of code that will be executed is code in the respective ISR. As mentioned above, all the interrupts will be disabled, the counters will freeze, the CUT
will pause its operation and another function is called that will inject the bit-flip on the FPGA based in the address that is written from the ARM core to the SEM core. After the injection of the fault, the time of the fault injection event is logged and the timers and the CUT resume execution. The AXI Timer 1 is loaded with the value of the next inter-arrival time from the DDR. It should be noted that, every delay in the process before injecting the bit-flip is taken into account and the actual inter-arrival times should be kept as close to the theoretical ones as possible. One source for introducing time delay, is the fast rate of scrubbing, that leads to a shift in time of the scheduled SEU injection. This is due to the fact that SEU injection and FPGA scrubbing are mutual exclusive processes in the scope of this study. Another source of delay is the time elapsed from the start of AXI Timer 1 in the ISR until the execution of the last code line of the AXI Timer’s 1 ISR. Finally, the third source of delay is due to the difference in time from the start of AXI Timer 0 until the start of AXI Timer 1 in the same ISR (few code lines below).

The AXI timer 2 is responsible to trigger a scrubbing event. For this reason, it is loaded with the same value during an experiment and is kept stable at a certain rate.

When a fault is going to be injected, the ARM pauses the CUT and resumes its execution after the fault is injected in the system. The reason for this is that the injection latency varies depending on the address of the bit flip. By pausing the execution of the CUT and the timers we emulate an instantaneous fault injection. When the fault injection process is completed, a time-stamp is logged and stored in the DDR3 memory. In the same way, after a scrubbing process completes, a time-stamp is logged as before. When the global timer reaches the amount of execution time that the user specifies, the interrupts are disabled and the ARM core sends all time-stamps back to the host PC for analysis.

4.3 SEU Framework Performance Evaluation

The metrics that are going to be used for the evaluation of the framework are: resources utilisation, SEU rate, scrubbing rate, the ratio of scrubbing rate over SEU rate \( \frac{f_{scrub}}{f_{seu}} \) and the error in time between the targeted and the actual fault injections.
Figure 4.10: Performance of the framework based on the original SEU distribution. The desired properties of the framework is to inject SEUs at different rates and with any distribution, perform scrubbing at certain rates and check whether the actual ratios of fscrub/fseu are the same with the targeted ones. Measurements regarding time are taken using the interrupts that are introduced in the system. The error between the actual and the targeted fault injections times is measured in seconds using the logged events from the interrupts of the AXI Timers.

The framework was synthesized and implemented on the ZYNQ platform. Table 4.2 indicates resource utilisation for all the main blocks of the framework. The design occupies 8.6% of the LUTs, 4.2% of the registers and 65% of the BRAMs of FPGA. Therefore, the low resource utilisation with respect to LUTs and slice registers, leaves 80% space for the CUT. However, the high BRAM utilisation is due to BRAMs used for debug purposes for the CUT and the gold design. If these BRAMs are removed, some extra BRAM resources will be available for more complex designs.

This study assumes an SEU rate of 1 SEU per second and if this rate decreases,
Table 4.2: Resources utilization of the framework on the ZYNQ without the CUT

<table>
<thead>
<tr>
<th></th>
<th>#LUTs</th>
<th># REGs</th>
<th># BRAMs</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEM core</td>
<td>891</td>
<td>1260</td>
<td>3</td>
</tr>
<tr>
<td>HWICAP</td>
<td>386</td>
<td>925</td>
<td>2</td>
</tr>
<tr>
<td>AXI4Lite</td>
<td>783</td>
<td>716</td>
<td>N/A</td>
</tr>
<tr>
<td>AXI Timers</td>
<td>1,011</td>
<td>648</td>
<td>N/A</td>
</tr>
<tr>
<td>User IP (without CUT)</td>
<td>1,471</td>
<td>879</td>
<td>87</td>
</tr>
<tr>
<td>Reset logic</td>
<td>23</td>
<td>31</td>
<td>N/A</td>
</tr>
<tr>
<td>Total Utilization</td>
<td>4,565</td>
<td>4,459</td>
<td>92</td>
</tr>
<tr>
<td>Available</td>
<td>53,200</td>
<td>106,400</td>
<td>140</td>
</tr>
<tr>
<td>Percentage(%)</td>
<td>8.6</td>
<td>4.2</td>
<td>65.7</td>
</tr>
</tbody>
</table>

fewer errors will be observed, but the actual time of the experiments will be proportionally longer. The execution time of one experiment was selected to be 30 minutes. Therefore, by having an SEU rate of 1 SEU/second, around 1,800 faults can be injected in half an hour.

The mean SEU rate of the environment the FPGA system will be operating in is assumed to be constant over time. The inter-arrival time of particles follows an exponential distribution having a mean that is the constant SEU rate. It should be noted that the framework can support any SEU injection distribution. In this work, we chose to use the SEU distribution from [94]. Because the system is dynamic there are cases that the assumed SEU rate and the actual one differ, due to the increase in the number of scrubs over a certain time period. In every experiment, the targeted SEU rate is compared with the actual SEU rate taken from the FPGA board, and this gives the error metric. Figure 4.10 presents a histogram of the difference between the actual and the targeted fault injection time versus the scrub over SEU rate versus the percentage of errors. It should be noted that increasing the scrub over SEU rate beyond 16 increases the error value between actual and targeted fault injections, deviating from the assumed exponential distribution. This happens because more scrubbing events are introduced in time resulting in time shifting of the fault injections.

In Figures 4.10 - 4.18, the error distribution of the actual versus theoretical fault
Figure 4.11: Error distribution of actual vs theoretical fault injection for fscrub/fseu = 1/16

injection can be found for the different fscrub rates. It is clear that as the scrubbing rate increases with the SEU rate being constant for all experiments, the mean and the standard deviation of the error increases leading to deviation from the original exponential distribution of the SEUs. Figure 4.11 illustrates the error histogram for the slowest scrubbing rate of 1 scrub every 16 seconds, while figure 4.19 demonstrates the distribution of error for highest rate of scrub which is 16 scrubs per second. The intermediate figures present the error histograms for a scrubbing rate that is double than the previous figure.
Figure 4.12: Error distribution of actual vs theoretical fault injection for fscrub/fseu = 1/8

Figure 4.13: Error distribution of actual vs theoretical fault injection for fscrub/fseu = 1/4
Figure 4.14: Error distribution of actual vs theoretical fault injection for $f_{scrub}/f_{seu} = 1/2$

Figure 4.15: Error distribution of actual vs theoretical fault injection for $f_{scrub}/f_{seu} = 1$
Figure 4.16: Error distribution of actual vs theoretical fault injection for fscrub/fseu = 2

Figure 4.17: Error distribution of actual vs theoretical fault injection for fscrub/fseu = 4
Figure 4.18: Error distribution of actual vs theoretical fault injection for fscrub/fseu = 8

Figure 4.19: Error distribution of actual vs theoretical fault injection for fscrub/fseu = 16
The mean and the standard deviation of the error histograms of Figures 4.11 - 4.19 are presented on Table 4.3. It can be concluded that as the scrubbing rate increases with the SEU rate kept constant higher errors are introduced, meaning that the framework is not following the targeted exponential distribution of the inter-arrival time of SEUs any more.

Table 4.3: Mean and standard deviation values for various \( f_{\text{scrub}}/f_{\text{seu}} \) ratios

<table>
<thead>
<tr>
<th>figure 4.8</th>
<th>fseu</th>
<th>fscrub</th>
<th>mean</th>
<th>std</th>
</tr>
</thead>
<tbody>
<tr>
<td>figure 4.9</td>
<td>1 seu/sec</td>
<td>1 scrub/16sec</td>
<td>5.552 × 10^{-6}</td>
<td>1.476 × 10^{-4}</td>
</tr>
<tr>
<td>figure 4.10</td>
<td>1 seu/sec</td>
<td>1 scrub/8sec</td>
<td>1.263 × 10^{-5}</td>
<td>2.397 × 10^{-4}</td>
</tr>
<tr>
<td>figure 4.11</td>
<td>1 seu/sec</td>
<td>1 scrub/4sec</td>
<td>1.267 × 10^{-5}</td>
<td>2.398 × 10^{-4}</td>
</tr>
<tr>
<td>figure 4.12</td>
<td>1 seu/sec</td>
<td>1 scrub/sec</td>
<td>3.467 × 10^{-5}</td>
<td>3.912 × 10^{-4}</td>
</tr>
<tr>
<td>figure 4.13</td>
<td>1 seu/sec</td>
<td>2 scrubs/sec</td>
<td>6.155 × 10^{-5}</td>
<td>5.273e-4</td>
</tr>
<tr>
<td>figure 4.14</td>
<td>1 seu/sec</td>
<td>4 scrubs/sec</td>
<td>1.212 × 10^{-4}</td>
<td>7.332e-4</td>
</tr>
<tr>
<td>figure 4.15</td>
<td>1 seu/sec</td>
<td>8 scrubs/sec</td>
<td>2.596 × 10^{-4}</td>
<td>1.1e-3</td>
</tr>
<tr>
<td>figure 4.16</td>
<td>1 seu/sec</td>
<td>16 scrubs/sec</td>
<td>4.69 × 10^{-4}</td>
<td>1.4e-3</td>
</tr>
</tbody>
</table>

4.4 Conclusion

This chapter presented a novel SEU mitigation framework on a Xilinx Zedboard that supports fault injection in the configuration memory of the FPGA through the SEM IP core and FPGA scrubbing using the HWICAP module. Experimental results show that it is a low area design occupying only 4.2% of the LUTs, 8.6% of the registers and 65% of the BRAMs on the Zedboard. The framework supports various SEU and scrub rates, but it is limited to the value of \( f_{\text{scrub}}/f_{\text{seu}} = 16 \), due to a mean error of \( 4.69 \times 10^{-4} \) and standard deviation 1.4e-3 between the targeted and the actual fault injections in time. This is due to the increased number of scrubs placed in time between SEU events, leading to a time shift of the SEUs and therefore, deviating from the targeted distribution. Effectively there is a trade-off between SEU rate, \( f_{\text{scrub}}/f_{\text{seu}} \) ratio and how long the experiment last. Therefore, selecting a lower SEU rate, increasing the scrubbing rate and running the experiments longer would decrease the values of errors between the original and the actual
SEU distributions because of the fewer SEUs. The framework designed and implemented in this chapter is going to be used for further experiments in the next chapter.
Chapter 5

Area-Driven Partial
Reconfiguration for SEU
Mitigation

5.1 Introduction

Scrubbing techniques such as blind scrubbing offer high availability to a design at the cost of excessive power overheads. On the other hand, an on-demand scrubbing method applied to a DMR circuit reduces the number of scrubs over time and thus power consumption and offers high availability at the cost of duplicating resources. An on-demand scrubbing technique applied to a design with TMR, introduces more than 200% area overheads and this could not be accepted by some applications.

This chapter presents two novel SEU mitigation schemes for fault-tolerant FPGA designs. The first scheme is based on DMR approach for detecting errors, but instead of replicating the whole design under protection, it uses the partial reconfiguration property of modern FPGA devices, and replicates only part of the design at any given time. Two techniques are applied: the round robin scheduling that cyclically checks for errors and the area-driven approach where checking is performed for time that is proportional to the size of the modules of the design.
The second scheme is an alternative to the on-demand scrubbing of a TMR design with the third replica accommodating only parts of the design at any given time for masking errors using dynamic partial reconfiguration. Round-robin scheduling and area-driven methods are also applied for the second scheme.

The underlying insight and intuition motivating the round-robin scheduling and area-driven techniques is that they will save circuit area because only part of the design is replicated as compared to the DMR or TMR. However, the proposed techniques will have lower availability when compared to on-demand scrubbing technique for DMR and TMR and this due to the fact that not all of the modules are being checked for errors at all times as in the case of DMR and TMR. Moreover, it is expected that both techniques will have smaller power overheads than blind scrubbing which brings the bitstream from an external memory for scrubbing at a very high rate to remove accumulated faults. In the cases of both round-robin scheduling and area-driven strategies, the scrubbing from the external memory is performed after a detection of an error and when the modules being checked are changing.

The aforementioned techniques can only be used for designs that form a feed-forward datapath and are state-less, such as FIR filters and other DSP filters. If the techniques are going to be used for circuits that contain feedback paths or the state of the circuit need to be saved, then other techniques such as check-pointing and rollback should be added.

The primary metrics of interest behind the proposed techniques is the availability of the system, the area and power consumption. The availability can be defined as the percentage of time that the design is operating correctly producing the expected outputs over the total time of operation. Blind scrubbing technique is applied to a baseline system, on-demand scrubbing is applied to a DMR or TMR design and the proposed techniques are applied to a novel reconfigurable design. Power consumption metric is the size of the data that are being read by an external memory for scrubbing. The most important metric when comparing on-demand scrubbing with the proposed technique is area versus availability and when we compare the proposed techniques with blind scrubbing the most important metrics are availability versus power consumption.
5.2 Baseline

Considering a design without any fault-tolerant feature in a radiation environment with a certain SEU rate, a scrubbing rate can be chosen to correct faults at the application level in order to not allow the accumulation of SEUs in the design. The faster the scrubbing rate, the more time the system is available but at the cost of power consumption. It should be noted that while scrubbing, the design is up and running, meaning that there is no loss of availability.

5.3 DMR system

An industry-standard technique for detecting errors is DMR with the addition of a comparator at the output of the two replicas that indicate any discrepancy. This design technique is applicable when there is extra space available on the chip to implement a redundancy scheme (i.e DMR). This structure gives information about the errors in the system and the error signal can be used as a trigger to perform scrubbing on demand as compared to the baseline where scrubbing is performed with a user-defined rate based on the expected SEU rate. This technique doubles the total area of the design, provides high availability and reduces the memory transfers for scrubbing since these transfers only happen after a detection of an error at the application level and not periodically as in the case of the baseline system.

5.4 Proposed SEU Mitigation Technique for a partial DMR system

In order to reduce the total required area of the CUT, a partial DMR technique is proposed in this work. Considering a modular component that can be partitioned into smaller modules, each one of the modules are being checked in a cyclic order instead of replicating the whole component. Two strategies are investigated based on the time at which each module is checked for errors. Figure 5.1 shows the round robin scheduling strategy where
the modules are checked for errors by being placed in Region 2. The time that each module stays in Region 2 is the same for all modules. Figure 5.2 demonstrates the area-driven strategy where the modules of the component are checked for errors for a time that is proportional to their area. Therefore, the module with the largest area is tested for errors for more time than the rest of the modules.
$t_1$ = time Region 2 is loaded with module 1
$t_2$ = time Region 2 is loaded with module 2
$t_3$ = time Region 2 is loaded with module 3
$t_4$ = time Region 2 is loaded with module 4

$t = t_1 = t_2 = t_3 = t_4$

Figure 5.1: Round robin scheduling
$a_1 = \text{area of module 1}$
$t_1 = \text{time Region 2 is loaded with module 1}$

$\text{module 1}$ $\text{module 2}$ $\text{module 3}$ $\text{module 4}$
Region 1

$\text{module 2}$ $\text{module 3}$ $\text{module 4}$
Region 2

$\text{module 1}$ $\text{module 2}$ $\text{module 3}$ $\text{module 4}$
Region 3

$\text{module 1}$ $\text{module 2}$ $\text{module 3}$ $\text{module 4}$
Region 4

$a_2 = \text{area of module 2}$
$t_2 = \text{time Region 2 is loaded with module 2}$

$a_3 = \text{area of module 3}$
$t_3 = \text{time Region 2 is loaded with module 3}$

$a_4 = \text{area of module 4}$
$t_4 = \text{time Region 2 is loaded with module 4}$

$\text{module 4}$ $\text{module 1}$ $\text{module 2}$ $\text{module 3}$
Region 2

$A = \text{total area} = a_1 + a_2 + a_3 + a_4$

$t_1 = \frac{a_1}{A} + a_1$

$t_2 = \frac{a_2}{A} + a_2$

$t_3 = \frac{a_3}{A} + a_3$

$t_4 = \frac{a_4}{A} + a_4$

time

Figure 5.2: Area-driven technique
5.5 Formal description of the method

5.5.1 Blind scrubbing on the baseline design

Figure 5.3 shows the block diagram of a modular design that consists of modules $i = 1...n$ with areas $a_1...a_n$. The total area of the design is the sum of the individual areas of the modules $\sum_{i=1}^{n} a_i$ from $a_1$ to $a_n$ and the maxin

![Block diagram of a modular design](image)

Figure 5.3: Block diagram of a modular design

The number of essential bits of the baseline design is defined as $\# \text{ essential bits of the baseline}$ and the $f_{\text{seu}}$ is the SEU rate that this design will operate measured in number of SEUs/second. Since the design does not have any detection mechanism, a scrub rate should be used to remove accumulated faults, defined as $f_{\text{scrub}}$ measured in number of scrubs/second. The availability of this design is defined as the percentage of time the design is operating correctly over the total time and is described by Equations 5.1, 5.2. Downtime is the time window from when we have an error in the output until the scrubbing is finished, and the design is corrected from faults.

$$\text{Availability (\%)} = \frac{\text{Uptime}}{\text{Total execution time}} \quad (5.1)$$

$$\text{Total execution time} = \text{Uptime} + \text{Downtime} \quad (5.2)$$

5.5.2 On-demand scrubbing on the DMR design

Figure 5.4 shows the DMR version of the generic baseline design able to support the on-demand scrubbing technique. The total area of the design is doubled:
Total\(_{DMR\text{area}}\) = 2 \* \(\sum_{i=1}^{n} a_i + a_{\text{comp}}\)

since the existed modules have been duplicated and a comparator is added that is checking the two output signals for a mismatch that would indicate a DMR error. The critical path and the \(F_{\max}\) have not changed since the extra hardware is operating in parallel.

Figure 5.4: Block diagram of a generic DMR design

The number of essential bits of the DMR design is doubled \(\# \text{ essential bits of the DMR} = 2^{*} \# \text{ essential bits of the baseline}\). Moreover, the SEU rate \(f_{SEU}\) of the DMR should be two times the \(f_{SEU}\) of the baseline design \((f_{SEU} \text{ of DMR} = 2 \* f_{SEU} \text{ of baseline})\), as it is proportional to the number of configuration bits and therefore proportional to the number of essential bits. The scrubbing is not based on a certain rate, but is triggered on-demand from a DMR error from the comparator of the DMR design. The mean time to repair \(MTTR\), that is the time needed for scrubbing the DMR design is doubled as the area of the DMR design is double the area of the baseline design.

The on-demand scrubbing technique is definitely increasing the required area of the system as compared to the blind scrubbing strategy applied on the baseline design but it offers high availability, since whenever a fault is manifested as an error, a scrubbing is performed to correct it. On the other hand, the blind scrubbing design is smaller in area and in order to achieve high availability, it needs to perform scrub at a very high rate which means that the power consumption is increased.
5.5.3 Proposed techniques on the reconfigurable DMR design

Figure 5.5 shows the proposed reconfigurable DMR design. The design is split in two reconfigurable partitions RP1, RP2. RP1 holds the original modules of the baseline with and two n-inputs multiplexers, one for providing the input to the module that is placed in the RP2 and the second for selecting one of the several outputs of the modules that is going to be compared with the output of the module in the RP2. For the aforementioned comparison, a comparator circuit is added signalling the DMR error. Therefore, the area of the reconfigurable DMR design is

\[
Total_{Reconf\, DMRArea} = \sum_{i=1}^{n} a_i + a_{comp} + 2 \times a_{mux} + max(a_1...a_n)
\]

Figure 5.5: Block diagram of a generic reconfigurable DMR design

The Reconfigurable DMR design will have less area than the DMR design if only the area of the multiplexers \(a_{mux}\) is smaller than the sum of the areas of the modules, excluding the maximum sized module:

\[
Total_{Reconf\, DMRArea} < Total_{DMRArea}
\]

\[
\Rightarrow \sum_{i=1}^{n} a_i + 2 \times a_{mux} + max(a_i...a_n) < 2 \times \sum_{i=1}^{n} a_i
\]

\[
\Rightarrow 2 \times a_{mux} + max(a_i...a_n) < \sum_{i=1}^{n} a_i
\]
\[ 2 \ast \alpha_{\text{max}} < \sum_{i=1}^{n} a_{i} - \max(a_{i}\ldots a_{n}) \]

In the case where we have many small size modules with more than one output to be checked for errors, then we will need more and larger in size multiplexers and therefore they will dominate the area of the design, then the DMR design could have less area than the reconfigurable design and will be a better choice. Therefore, the proposed design saves area when compared to a DMR when it consists of a few number of large area modules and small size multiplexers.

The critical path and the \( F_{\text{max}} \) of the reconfigurable DMR design is the same with the critical path and the \( F_{\text{max}} \) of the baseline and the DMR. The reconfigurable DMR design uses two scrubbing techniques: round robin scheduling and area-driven reconfiguration that are presented below.

**Round robin scheduling**

First of all, we need to calculate the SEU rate for the round robin scheduling technique. In order to do that, the number of essential bits of the reconfigurable design should be calculated first. Since the design consists of two RPs, the number of essential bits of RP1 should be added to the number of essential bits of the RP2. The number of essential bits of RP1 is greater than the number of essential bits of the baseline because of the added multiplexers.

\[ \text{EB (Round robin RP1)} > \text{EB (baseline)} \]

Because the round robin scheduling uses constant rate of changing the modules in the RP2, the number of essential bits of the RP2 is the average of the essential bits of each module:

\[ \text{EB (Round robin RP2)} = \frac{\sum_{i=1}^{n} EB_{i}}{n} \]


\[
EB \text{ (Round robin)} = EB \text{ (Round robin RP1)} + EB \text{ (Round robin RP2)}
\]

\[
\Rightarrow EB \text{ (Round robin)} = EB \text{ (Round robin RP1)} + \frac{\sum_{i=1}^{n} EB_i}{n}
\]

Therefore the SEU rate for the round robin scheduling is:

\[
f_{seu} \text{ (round-robin)} = f_{seu} \text{ (baseline)} * \frac{EB \text{ (Round robin)}}{EB \text{ (baseline)}}
\]

Based on the \( f_{seu} \), we can define various scrubbing rates \( f_{scrub} \) in order to compare the availability of the system with the baseline design for the same ratios of \( \frac{f_{scrub}}{f_{seu}} \).

The \( MTTR \) for the reconfigurable DMR design implementing the round robin scheduling technique depends on the size of the \( n \) modules. A small sized module will have small \( MTTR \) and a module that has large area will have large \( MTTR \). The assumption is that the targeted FPGA will not have a frame structure in place, so the previous argument holds.

\[
MTTR \text{ (Round robin)} = MTTR \text{ (Round robin RP1)} + \frac{\sum_{i=1}^{n} MTTR_i}{n}
\]

The availability metric depends on both how fast we detect an error and how fast we reconfigure to recover from the error. In other words, the lowest the downtime of the system the higher the availability will be. The detection of the error using the round robin scheme depends on how fast we switch the modules if the fault was on a different module than the one that we were checking. By changing fast the modules cyclically, ultimately we will catch the error and we will scrub both RP1 and RP2 to recover from that. However, when the modules are heterogeneous and we have small and large modules, it is more likely that the large modules will be more sensitive to faults and by not spending more time checking it than the rest of the modules we might be slow in detecting an error and therefore, the design will have lower availability.
Area-driven reconfiguration

An alternative to the round robin scheduling approach is the area-driven technique that takes into account area information in order to check for time that is proportional to the area of the modules of the design. The metrics for the reconfigurable DMR design with the area-driven method will not be the same. The number of essential bits of the RP2 is a weighted average of the essential bits of the modules of the design. Thus, we assume that we have a set of weights $w_1...w_n$ that they should have the following property $\sum_{i=1}^{n} w_i = 1$. Every weight is calculated as

$$w_i = \frac{EB_i}{EB_i + EB_{RP1}}$$

The total number of essential bits of this approach is:

$$EB (area-driven RP2) = \sum_{i=1}^{n} w_i \cdot EB_i$$

$$EB (area-driven) = EB (area-driven RP1) + EB (area-driven RP2)$$

$$\implies EB (area-driven) = EB (area-driven RP1) + \sum_{i=1}^{n} w_i \cdot EB_i$$

Therefore the SEU rate for the area-driven technique is calculated below:

$$f_{seu} (area-driven) = f_{seu} (baseline) \cdot \frac{EB (area-driven)}{EB (baseline)}$$

Having a different $f_{seu}$ rate than the round robin requires different scrubbing rate $f_{scrub}$ for changing the modules in the RP2 region in order to keep the same the ratio $\frac{f_{scrub}}{f_{seu}}$ and make the comparisons with the other designs.

The $MTTR$ is the sum of the $MTTR$ of the RP1 plus the weighted average of the
$MTTR$ of the RP2 as shown below:

$$MTTR \ (area-driven) = MTTR \ (RP1) + \sum_{i=1}^{n} w_i \ast MTTR_i$$

The availability of the reconfigurable DMR with the area-driven technique will be different than the availability of the round robin scheduling due to the change in the $MTTR$ and the $f_{seq}$ and $f_{scrub}$. Therefore the downtime will be different, and the prediction is that the results will be better as more faults occur in the large area modules and checking them more time we will detect errors faster.

### 5.6 Second-Order Polynomial Case Study

The case study used in this work is a polynomial evaluation function

$$F(x) = 3x^2 + 5x + 5 \quad (5.3)$$

where the input $x$ is 16-bits. The reason this case study was selected is because it can be partitioned in its arithmetic operators for protection and it is a feed-forward datapath. In addition, the selected case study has common operators found in signal processing applications. Figure 5.6 presents the block diagram of the aforementioned function. It consists of two adders, one constant coefficient multiplier and a two-input multiplier that occupies most of the area of the entire circuit. All operators support integer arithmetic with wordlength of 16-bits. The blind scrubbing technique is applied on this design, since it does not have any fault-tolerant feature.

Figure 5.7 demonstrates the DMR version of the baseline where the entire component is duplicated and a comparator is added for detecting an error. This design is used to evaluate the on-demand triggered scrubbing technique. The proposed solution is illustrated in Figure 5.8. The design is extended with multiplexers, a comparator that signals a mismatch and a reconfigurable partition for instantiating any part of the design for protection at any given time.
Figure 5.6: Baseline Polynomial

Figure 5.7: DMR Polynomial
5.7 High level system operation

The baseline, DMR and Reconfigurable DMR versions of the second-order polynomial are integrated in the framework presented in Chapter 4. The input data are sent from the ARM CPU to the external memory of the framework and in turn to the local input BRAM of the CUT. Both CUT and gold designs read the same data cyclically from the input BRAM and they should produce the same outputs. A comparator signals a mismatch between the CUT and gold outputs in case of an error due to a fault injected. This error signal is connected to the ”oracle” interrupt which is only there for testing purposes. The three AXI timers that are connected to the AXI4lite bus generate interrupts that trigger the data logging for timing events. AXI timer 0 counts the whole duration of every experiment while AXI Timer 1 is responsible to inject faults at predefined time intervals based on the assumed SEU distribution. AXI Timer 2 is used to trigger the scrubbing process at specified intervals.

The baseline system is placed in one RP on the FPGA and does not have any fault detection capability. The only interrupts that are used when this system is up and running
are the three interrupts coming from the AXI timers and the oracle interrupt indicating
the error between the baseline and the gold design. Fault injection and FPGA scrubbing
are performed only in the RP where the baseline is placed, while the rest of the framework
is operating.

The DMR design is also placed in one RP and has an internal comparator that signals
an error between the redundant datapaths and is connected to the DMR error interrupt,
that triggers the on-demand scrubbing process. Therefore, when the DMR system is
integrated in the framework four interrupts are used. The AXI Timer 2 interrupt that
schedules scrubs at regular intervals is disabled in this case. A new interrupt is enabled
that indicated a DMR error between the two redundant second-order polynomials and
triggers the on-demand scrubbing. Fault injection and FPGA scrubbing are performed
only in the RP where the DMR is placed, while the rest of the framework is operating.

The Reconfigurable DMR system is split in two RPs, with one having the original
second-order polynomial plus the extra logic and the multiplexers for routing signals to
RP2 where modules are loaded based on one of the two scrubbing strategies (round robin
scheduling or area-driven technique). Five interrupts are used, when the Reconfigurable
DMR system is connected to the framework. The AXI timer 2 is used to perform scrubs
at regular intervals in RP2, by loading the operators cyclically. The difference with the
DMR design is that the comparator triggers the DMR interrupt when it detects errors
between the modules that are being checked. It should be noted that fault injection and
FPGA scrubbing are performed in the two RPs that the Reconfigurable DMR design is
placed.

5.8 Performance Evaluation of FPGA scrubbing techniques
(DMR)

The aforementioned designs with the corresponding scrubbing strategies were evaluated
using the framework that was described in the previous chapter. The metrics used for eval-
uating the existing scrubbing techniques and the proposed one are: availability, MTTR,
size of data transferred from the memory and area.

5.8.1 SEU rate

The targeted operation environments are space application and scientific laboratories such as CERN with particle accelerators. The SEU rates for upsets in the configuration memory of a space qualified Xilinx XQR4VSX55 FPGA device for two orbits were modelled using the CREME model [95] in [25]. The authors reported a SEU rate of 7.56 upsets/device*day for the LEO orbit and 4.28 upsets/device*day for the GEO orbit. Brosser et. al [73] used for their experiments a SEU rate of 1 SEU per minute targeting space application. However, injecting thousands of faults with such a rate would require long experiment runs. In this work, a SEU rate of 1 SEU/sec is selected in order to accelerate testing.

The SEU rates are calculated based on the number of essential bits per unit area of each design. In this work, we consider as unit area an essential configuration bit. Table 5.1 summarises the number of essential bits and the SEU rates for the baseline, the DMR and the uniform and area-driven techniques for the reconfigurable DMR designs.

Table 5.1: SEU rates and number of essential bits for Baseline, DMR, uniform and area-driven scrubbing techniques

<table>
<thead>
<tr>
<th>Partitions</th>
<th>Number of essential bits</th>
<th>SEU rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>RP</td>
<td>78,666</td>
</tr>
<tr>
<td>(Blind scrubbing)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DMR</td>
<td>RP</td>
<td>162,806</td>
</tr>
<tr>
<td>(On-demand scrubbing)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Round robin scheduling</td>
<td>RP 1</td>
<td>82,225</td>
</tr>
<tr>
<td></td>
<td>RP 2</td>
<td>10,310 (average)</td>
</tr>
<tr>
<td>Area driven</td>
<td>RP 1</td>
<td>82,225</td>
</tr>
<tr>
<td></td>
<td>RP 2</td>
<td>14,887 (weighted average)</td>
</tr>
</tbody>
</table>

The baseline design has 78,666 essential bits and the SEU rate is assumed to be 1 SEU per second per unit area. The rest SEU rates are derived based on the SEU rate of the baseline and the number of essential bits. The DMR design with the on-demand scrubbing strategy has 162,806 essential bits and the SEU rate used when experimenting
with the DMR design is calculated by Equation 5.4.

\[
SEU \text{ rate of the DMR} = \frac{\# \text{ essential bits of the DMR}}{\# \text{ essential bits of the baseline}} \times \text{SEU rate of the baseline}
\]

\[
= \frac{162,806}{78,666} = 2.07
\]

The Round robin scheduling and area-driven techniques are evaluated under different SEU rates. This is because they have a different number of essential bits. It should be noted that the reconfigurable DMR design is implemented in two reconfigurable partitions. RP1 includes the second order polynomial with the added functionality and RP2 is the partition where operators can be loaded on-demand. The Round robin scheduling assumes that the time each operator is placed on the RP2 for checking for errors is identical. Four modules can be placed on RP2: one of the two identical adders, the constant coefficient multiplier or the two-inputs multiplier. The number of essential bits of the modules that can be placed on the RP2 are shown in Table 5.2.

<table>
<thead>
<tr>
<th>Module</th>
<th># essential bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>6,072</td>
</tr>
<tr>
<td>MULT1</td>
<td>6,072</td>
</tr>
<tr>
<td>MULT2</td>
<td>23,023</td>
</tr>
</tbody>
</table>

The number of essential bits of the RP2 on average is calculated based on Equation 5.5.

\[
RP2 = \frac{2 \times ADD + \text{MULT1} + \text{MULT2}}{4} = 10,310
\]

The SEU rate of the round robin scheduling is calculated based on the fraction of the
total number of essential bits over the number of essential bits of the baseline according to Equation 5.6.

\[
SEU \text{ rate of the round robin scheduling} = \frac{RP1 + RP2}{\# \text{ essential bits of the baseline}} \times SEU \text{ rate of the baseline} \tag{5.6}
\]

\[
= 1.18
\]

For the area-driven scrubbing technique, a weighted average of the number of essential bits of the RP2 partition is computed. The weights are computed based on the percentage of essential bits of each operator and the total number of essential bits, when the same operator is placed in the RP2 partition, as shown in Equation 5.7. It should be mentioned that the weight of ADD operator and the weight of MULT1 operator are equal since they both include the same number of essential bits.

\[
\text{Weight1} = \text{Weight}_{\text{MULT1}} = \text{Weight}_{\text{ADD}} = \frac{ADD}{RP1 + ADD} = 0.069 \tag{5.7}
\]

Equation 5.8 computes the weight of the two-input multiplier, that is the largest module in terms of area of the polynomial IP.

\[
\text{Weight2} = \text{Weight}_{\text{MULT2}} = \frac{MULT2}{RP1 + MULT2} = 0.22 \tag{5.8}
\]

The two weights should sum up to 1, since this would be an entire cycle through the operators. However, the addition of weight1 (multiplied by 3 for the two adders and the constant coefficient multiplier) and weight 2 equals 0.43. Therefore, the sum of the two weights is scaled to 1 as shown in Equation 5.9.
NewWeight1 = \frac{0.069 \times 100 \%}{0.43} = 16\% \quad (5.9)

NewWeight2 = \frac{0.22 \times 100 \%}{0.43} = 52\%

Thus, according to Equation 5.9, the weighted average of the essential bits of the RP2 for the area-driven technique is shown in Equation 5.10

\[ Rp2 = 2 \times 0.16 \times ADD + 0.16 \times MULT1 + 0.52 \times MULT2 = 14,887 \quad (5.10) \]

Finally, Equation 5.11 computes the SEU rate of the area-driven scrubbing technique for the reconfigurable DMR polynomial.

\[
SEU \, rate \, of \, the \, area-driven = \frac{RPI + RP2}{\# \, essential \, bits \, of \, the \, baseline} \times SEU \, rate \, of \, the \, baseline
\]

\[ = 1.23 \]

The ratio of scrubbing rate over SEU rate (fscrub/fseu) should be kept the same for the baseline, the DMR and the reconfigurable DMR designs. Table 5.3 represents the scrubbing rates and SEU rates for the round robin scheduling and the area-driven scrubbing techniques that were chosen for this study. The motivation behind the choice of such fscrub/fseu ratios is that these ratios were used in the baseline polynomial when we were examining the performance capabilities of the framework presented in Chapter 4. Moreover, we wanted to investigate the impact of having lower or higher scrubbing rate on the availability of the system and the total amount of data that is transferred from the external memory for scrubbing. Finally, similar scrubbing rates were used by [73] for evaluating the robustness of various scrubbing rates on the FIT of various designs.
Table 5.3: fscrub/fseu, SEU and scrubbing rates for the round robin scheduling and area-driven techniques

<table>
<thead>
<tr>
<th></th>
<th>fscrub/fseu</th>
<th>fscrub(event/sec)</th>
<th>fseu (event/sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Round robin scheduling</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/16</td>
<td>0.07375</td>
<td>1.18</td>
<td></td>
</tr>
<tr>
<td>1/8</td>
<td>0.1475</td>
<td>1.18</td>
<td></td>
</tr>
<tr>
<td>1/4</td>
<td>0.295</td>
<td>1.18</td>
<td></td>
</tr>
<tr>
<td>1/2</td>
<td>0.59</td>
<td>1.18</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1.18</td>
<td>1.18</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2.36</td>
<td>1.18</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4.72</td>
<td>1.18</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>9.44</td>
<td>1.18</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>18.88</td>
<td>1.18</td>
<td></td>
</tr>
<tr>
<td><strong>Area-driven</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/16</td>
<td>0.076875</td>
<td>1.23</td>
<td></td>
</tr>
<tr>
<td>1/8</td>
<td>0.15375</td>
<td>1.23</td>
<td></td>
</tr>
<tr>
<td>1/4</td>
<td>0.3075</td>
<td>1.23</td>
<td></td>
</tr>
<tr>
<td>1/2</td>
<td>0.615</td>
<td>1.23</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1.23</td>
<td>1.23</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2.46</td>
<td>1.23</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4.92</td>
<td>1.23</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>9.84</td>
<td>1.23</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>19.68</td>
<td>1.23</td>
<td></td>
</tr>
</tbody>
</table>

5.8.2 Availability

The availability is the percentage of time the design is operating producing correct results over the total time of operation as described by Equations 5.1 and 5.2. Downtime is the amount of time from when an error occurred until the completion of the scrubbing. The time when the error occurred is logged by the oracle interrupt which is the interrupt connected to the comparator output of the CUT and the Golden outputs.

Figure 5.9 demonstrates the availability of different techniques for several ratios of scrub over SEU rate. The y-axis includes the confidence intervals of the availability for the experiments that were run and the x-axis includes the blind scrubbing technique, the on-demand scrubbing technique, the round robin scheduling and the area-driven scrubbing technique for the fscrub/fseu ratios from Table 5.3. The same ratios were used for the
blind scrubbing technique in order to have a fair comparison. However, the one-demand scrubbing technique does not use scrubbing with a certain rate, but scrubbing is triggered only after the detection of a DMR error (DMR error interrupt). As we can see from Figure 5.9, "Blind (fscrub/fseu = 16)" achieves the maximum value of availability and this is due to the very high number of scrubs of the RP where the baseline is placed. Moreover, "Blind (fscrub/fseu = 1/16)" has the lowest availability between all the techniques and the reason behind this is the small scrubbing rate used in this case that leads to accumulated faults in the baseline system. The on-demand scrubbing technique achieves high availability without having scrubs at certain rates, but scrubbing is triggered only after the detection of an error. Round robin scheduling and area-driven techniques perform better in terms of availability than blind scrubbing for small ratios of fscrub/fseu. However, when fscrub/fseu is increased blind scrubbing techniques achieve higher availability than the proposed techniques but as we will see in the following sections at the cost of high power consumption due to very large amount of data transferred from external memory to scrub the entire RP where the baseline resides.
Figure 5.9: Confidence intervals of availability for the blind, on-demand, round robin scheduling and area-driven scrubbing techniques
5.8.3 Area Resources

Table 5.4 shows area results for the three systems when they are implemented on the ZYNQ platform. The baseline occupies 508 Lookup Tables (LUTs) and 114 Registers (REGs) while the DMR’s resources utilisation is 1057 LUTs and 230 REGs. The reconfigurable DMR saves 43.6% LUTs and 40.9% REGs as compared to DMR when the round robin scheduling is in place, and 40.2% LUTs and 40.9% REGs when the area-driven technique is used.
Table 5.4: Resources utilisation of the three designs

<table>
<thead>
<tr>
<th></th>
<th>RPs</th>
<th>#LUTs</th>
<th># REGs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Baseline</strong></td>
<td>RP</td>
<td>508 out of 1600 (31.75%)</td>
<td>114 out of 3200 (3.56%)</td>
</tr>
<tr>
<td><strong>DMR</strong></td>
<td>RP</td>
<td>1057 out of 1600 (66%)</td>
<td>230 out of 3200 (7.2%)</td>
</tr>
<tr>
<td><strong>Reconf. DMR</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RP1</td>
<td>530 out of 1600 (33.12%)</td>
<td>120 out of 3200 (3.75%)</td>
</tr>
<tr>
<td></td>
<td>RP2(ADD)</td>
<td>32 out of 1600 (2%)</td>
<td>16 out of 3200 (0.5%)</td>
</tr>
<tr>
<td></td>
<td>RP2(MULT1)</td>
<td>32 out of 1600 (2%)</td>
<td>16 out of 3200 (0.5%)</td>
</tr>
<tr>
<td></td>
<td>RP2(MULT2)</td>
<td>166 out of 1600 (10.38%)</td>
<td>16 out of 3200 (0.5%)</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td>596 out of 1600 (37.21%)</td>
<td>136 out of 3200 (4.3%)</td>
</tr>
<tr>
<td></td>
<td>(round robin scheduling)</td>
<td>43.6% saving over DMR</td>
<td>40.9% saving over DMR</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td>632 out of 1600 (39.5%)</td>
<td>136 out of 3200 (4.3%)</td>
</tr>
<tr>
<td></td>
<td>(area-driven)</td>
<td>40.2% saving over DMR</td>
<td>40.9% saving over DMR</td>
</tr>
</tbody>
</table>
5.8.4 Mean Time to Repair (MTTR)

The definition of MTTR is the average time needed to repair a faulty system. In order to calculate the MTTR of the three systems we need to take into account their reconfiguration times. The assumption made is that no frame structure is in place. The baseline polynomial is implemented in a reconfigurable partition that takes 1,600 LUTs and 3200 REGs, which is 58,5836 bytes. The time needed to reconfigure this partition from the DDR3 memory via the ICAP port is 7msec using a 50MHz clock frequency.

The resources utilisation should be taken into account in order to calculate the MTTR of each design. Therefore, Equations 5.12, 5.13 compute the MTTR of each design.

\[
MTTR_{Baseline} = 7 \text{ msec} \times (\% \text{ Baseline LUTs} + \% \text{ Baseline REGs}) = 2.47 \text{ msec} \quad (5.12)
\]

\[
MTTR_{DMR} = 7 \text{ msec} \times (\% \text{ DMR LUTs} + \% \text{ DMR REGs}) = 5.124 \text{ msec} \quad (5.13)
\]

The reconfigurable DMR consists of two identical in size RPs, as compared to the baseline and the DMR designs that only include one RP. The RP2 of the reconfigurable design accommodates one of the four operators of the polynomial and based on the scrubbing strategy, two MTTR should be computed.

The MTTR of the RP1 is identical for the round robin scheduling and area-driven techniques and calculated as shown in Equation 5.14.

\[
MTTR_{RP1} = 7 \text{ msec} \times (\% \text{ RP1 LUTs} + \% \text{ RP1 REGs}) = 2.58 \text{ msec} \quad (5.14)
\]

The MTTR of the adder module or the constant coefficient multiplier is computed in Equation 5.15, while the MTTR the two-input multiplier is presented in Equation 5.16.
\[ MTTR_{ADD} = 7 \text{ msec} \times (\% \text{ ADD LUTs} + \% \text{ ADD REGs}) = 0.175 \text{ msec} \quad (5.15) \]

\[ MTTR_{MULT2} = 7 \text{ msec} \times (\% \text{ MULT2 LUTs} + \% \text{ MULT2 REGs}) = 0.761 \text{ msec} \quad (5.16) \]

The MTTR of the Round robin scheduling technique is calculated by the average of the reconfiguration times of the four operators, depicted in Equation 5.17.

\[ MTTR_{\text{round robin scheduling}} = MTTR_{RP1} + \frac{3 \times MTTR_{ADD} + MTTR_{MULT2}}{4} = 2.9 \text{ msec} \quad (5.17) \]

The MTTR of the area-driven technique is calculated by the Equation 5.18 based on the weights that have been calculated earlier in Equation 5.9.

\[ MTTR_{\text{area-driven}} = MTTR_{RP1} + 3 \times 0.16 \times MTTR_{ADD} + 0.52 \times MTTR_{MULT2} = 3.06 \text{ msec} \quad (5.18) \]

Table 5.5 illustrates the MTTR for the three designs. The fastest MTTR belongs to the baseline and the slowest to the DMR design. Both versions of the Reconfigurable DMR offer smaller MTTR than DMR because they occupy less resources and higher MTTR as compared to the baseline because they occupy more area. It is very important for the availability of a system to have small MTTR as it indicates the ability of the system to recover from a fault and produce correct results without sacrificing availability. Therefore, the smaller this metric is, the better the availability of a fault-tolerant system would be.
Table 5.5: MTTR for the baseline, the DMR and the Reconfigurable DMR designs

<table>
<thead>
<tr>
<th></th>
<th>MTTR in msec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>2.47</td>
</tr>
<tr>
<td>DMR</td>
<td>5.124</td>
</tr>
<tr>
<td>Reconfigurable DMR</td>
<td></td>
</tr>
<tr>
<td>(round robin scheduling)</td>
<td>2.9</td>
</tr>
<tr>
<td>Reconfigurable DMR</td>
<td>3.06</td>
</tr>
<tr>
<td>(area-driven)</td>
<td></td>
</tr>
</tbody>
</table>

5.8.5 Size of data transferred from DDR3 for scrubbing

The size of data transferred from DDR3 depends on the size of the bitstream and on the number of scrubs that are performed to correct SEUs. Table 5.6 includes the bitstream size for each one of the three designs. The baseline bitstream size is 20,776 bytes while the size of the bitstream for the DMR is 43,068 bytes. Finally, the reconfigurable DMR design has 21,693 bytes for the RP1 partition and 1,471 or 6,402 bytes on RP2 depending on the operator that is loaded. The methodology for calculating the bitstream size of the baseline, the DMR and the reconfigurable DMR designs is the same as the one followed in computing the MTTR.

Table 5.6: Bitstream size in bytes for the three designs

<table>
<thead>
<tr>
<th></th>
<th>RPs</th>
<th>Bitstream size in bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>RP</td>
<td>20,776</td>
</tr>
<tr>
<td>DMR</td>
<td>RP</td>
<td>43,068</td>
</tr>
<tr>
<td>Reconf. DMR</td>
<td>RP1</td>
<td>21,693</td>
</tr>
<tr>
<td></td>
<td>RP2(ADD)</td>
<td>1,471</td>
</tr>
<tr>
<td></td>
<td>RP2(MULT1)</td>
<td>1,471</td>
</tr>
<tr>
<td></td>
<td>RP2(MULT2)</td>
<td>6,402</td>
</tr>
</tbody>
</table>

Figure 5.10 illustrates the trade-off between unavailability and size of data transferred from DDR3 per technique, having the origin of the diagram as the best performing point. It should be noted that the large amount of data transferred from DDR corersponds to a high power consumption due to the continous activity of the scrubbing circuit.
The best design point based on the aforementioned trade-off is the on-demand scrubbing strategy applied on a DMR version of the second-order polynomial design because it achieves very high availability and small size of data transferred from external memory and therefore, low power consumption due to scrubbing. However, the DMR design with the on-demand scrubbing method has double the area of the baseline design. The blind scrubbing technique applied to the baseline design achieves very high availability as we increase the scrubbing rate but at the cost of consuming more power for scrubbing due to the large amount of data transferred from the external memory. The area-driven scrubbing technique outperforms the round robin scheduling and this is due to the fact that it uses area information as described earlier to check for errors. For mid-range ratios of fsrub/fseu the area-driven method performs better on the trade-off than blind scrubbing by saving power consumption due to a decreased amount of data that have to be transferred for scrubbing. Therefore, dynamic partial reconfiguration is an effective technique for scrubbing saving area as compared to DDR and power consumption for scrubbing as compared to blind scrubbing with a small drop in availability.
5.9 Validation of the formal model (DMR)

The formal model developed in section 5.5 will be applied to second order polynomial function. An attempt to compare predicted with empirical data will be presented. The assumptions applied in the model are:

- The inter arrival time of SEUs are taken from an exponential distribution
- Only 20% of the faults lead to an error visible to the system output
- The SEU rate of the baseline is $f_{seu} = 1$
• The total duration of the tests are set to 30 minutes

Figure 5.11 presents the comparison of the availability results of the polynomial case study based on the formal model and results taken from experiments on the ZYNQ platform. From left to right, the first method that is being compared is blind scrubbing. The first three availability results are taken from the formal model for trying different values of the scrubbing rate, from \( f_{\text{scrub}} = 1/32 \) to \( f_{\text{scrub}} = 1/128 \). We observe that the availability of the baseline case study using the blind scrubbing at very low scrubbing rates as compared to the SEU rates is almost zero.

The points on the x-axis from (4 ...31) with interval 1 correspond to the model while the points from (4.5 - 31.5) with interval 1 correspond to the results taken from the ZYNQ platform. The availability results from the two methods for the blind scrubbing and the on-demand scrubbing are very close to each other. This is due to the assumption made in the model that as soon as an error appears in the system, it will be detected after a certain time interval. for the round-robin and the area-drive methods, the results from the model slightly differ than the experimental results due to the unknown time of detection of the error due to the continuous change of the modules in the RP2. The most clear variation between the model and the actual results, is in the round robin technique and especially in the low rates of scrub. This is probably due to the assumption that the error will be detected at the worst case scenario which is at the end of checking the module. So for example if we change the modules every 1 second, we assume that the detection time interval of a fault will be 1 second. However in reality, the fault might be detected earlier or later if the fault is another module that will be loaded later on for checking.
Figure 5.11: Comparison of the formal model with the actual data.
5.10 Formal description of the proposed method for a TMR system

5.10.1 On-demand scrubbing on a TMR design

Figure 5.12 shows the block diagram of a generic TMR system that is partitioned into 4 RPs. The main datapath is triplicated and each one of the three replicas occupy one RP. The fourth RP includes the voter circuitry that masks a single error and identifies which of the three replicas is in error. The total area of such a design is given by the following formula:

\[ \text{Total area} = 3 \sum_{i=1}^{n} a_i + a_{voter} \]

\[ \implies \text{Total area} = 3 \sum_{i=1}^{n} a_i + 3 \cdot a_{and} + 3 \cdot a_{xor} + a_{or} \]

The number of essential bits of the TMR system will be:

\[ EB \ (TMR) = EB \ (RP1) + EB \ (RP2) + EB \ (RP3) + EB \ (Voter) \]

Therefore the SEU rate \( f_{seu} \) of the TMR design should be:

\[ f_{seu} \ (TMR) = f_{seu} \ (baseline) \cdot \frac{EB \ (TMR)}{EB \ (baseline)} \]

Scrubbing is occurring on demand when any of the error signals of Figure 5.12 is set to a logic ‘1’ value. Thus, only the faulty replica and the voter circuit will be scrubbed. The \( MTTR \) of the TMR system is given by the following formula:

\[ MTTR \ (TMR) = MTTR \ (RP1=RP2=RP3) + MTTR \ (Voter) \]

The \( MTTR \) of the voter is the same for any design that consist of \( n \) modules that are being connected with each other and is independent of the number of modules \( n \) and the areas
of the modules. The $MTTR$ of RP1 or RP2 or RP3 (they are the same) is proportional to the area of the design that is triplicated.

The TMR design implementing the on-demand scrubbing is expected to have very high availability since it can mask a single error in one of the three replicas and it only needs to be scrubbed when there is a fault in the voter circuit.
Figure 5.12: Generic TMR system
5.10.2 Proposed techniques for a reconfigurable TMR system

Figure 5.13 shows the proposed reconfigurable TMR design that consists of two identical replicas (RP1, RP2) where $i = j = 1$ and $n = k$ and area $a_i = a_j$, a replica that includes the voter and the reconfigurable replica that can accommodate any one of the modules $i...n$, $j...n$. The design has $2n$ number of 2-input multiplexers, 4 $n$-input multiplexers, 5 XOR gates, 3 AND gates and 1 OR gate. The total area of the reconfigurable TMR design is:

\[
\text{Total area} = 2 \sum_{i=1}^{n} a_i + a_{\text{voter}} + \max(a_i...a_n)
\]

\[
\implies \text{Total area} = 2 \sum_{i=1}^{n} a_i + 2 \cdot n \cdot a_{\text{max2input}} + 4 \cdot a_{\text{muxninput}} + 5 \cdot a_{\text{xor}} + 3 \cdot a_{\text{and}} + a_{\text{or}} + \max(a_i...a_n)
\]

The reconfigurable TMR design has less area than the TMR only if:

\[
\text{Total}_{\text{Reconf TMR area}} < \text{Total}_{\text{TMR area}}
\]

\[
\implies \sum_{i=1}^{n} a_i > 2 \cdot n \cdot a_{\text{max2input}} + 4 \cdot a_{\text{muxninput}} + 2 \cdot a_{\text{xor}} + \max(a_i...a_n)
\]

\[
\implies \sum_{i=1}^{n} a_i - \max(a_i...a_n) > 2 \cdot n \cdot a_{\text{max2input}} + 4 \cdot a_{\text{maxninput}} + 2 \cdot a_{\text{xor}}
\]

Therefore, when the design has larger area modules than the size of the multiplexers the proposed solution saves area as compare to the TMR. Therefore a design that will be designed using this technique should be partitioned in large area modules so that the sum of the area of the modules will be greater than the multiplexer overheads. Moreover, a design with a large number of modules will need large $n$-input multiplexers and the area overheads of these multiplexers should be less than the total area of the modules of the design. In a case where, the design consists of multiple small sized modules having less area than the area of the multiplexers and a very large module, the TMR design may have less area than the proposed.
Figure 5.13: Generic Reconfigurable TMR system
**Round robin technique**

In order to calculate the SEU rate for the round robin scheduling technique, the number of essential bits of the reconfigurable TMR design should be calculated first. Since the design consists of four RPs, the number of essential bits of all RPs should be added. But, since we are using the round robin technique, the reconfigurable RP (RP4) will have the average number of essential bits of the modules that are placed in it. Therefore:

$$EB\ (Round\ robin\ RP4) = \frac{\sum_{i=1}^{n} EB_i}{n}$$

$$EB\ (Round\ robin) = EB\ (Round\ robin\ RP1) + EB\ (Round\ robin\ RP2) +$$

$$EB\ (Round\ robin\ Voter) + \frac{\sum_{i=1}^{n} EB_i}{n}$$

So the SEU rate for the round robin scheduling of the reconfigurable TMR is:

$$f_{seu}\ (round-robin) = f_{seu}\ (baseline) \ast \frac{EB\ (Round\ robin)}{EB\ (baseline)}$$

Based on the $f_{seu}$ which is larger than the baseline since the reconfigurable TMR design has greater number of essential bits than the baseline, we can define various scrubbing rates $f_{scrub}$ in order to compare the availability of the system with the baseline design for the same ratios of $\frac{f_{scrub}}{f_{seu}}$. The scrubbing rate gives as the rate that we load operators in the RP4 for checking. The correction of a fault is triggered on-demand when there is an error detected by one of the XOR gates as shown in Figure 5.13.

The $MTTR$ for the reconfigurable TMR design implementing the round robin scheduling technique depends on the size of the $n$ modules. A small sized module will have small $MTTR$ and a module that has large area will have large $MTTR$. The assumption is that the targeted FPGA will not have a frame structure in place, so the previous argument holds.
MTTR (Round robin) = MTTR (Round robin RP1) + MTTR (Round robin RP2) +

\[
MTTR (Round robin Voter) + \sum_{i=1}^{n} \frac{MTTR_i}{n}
\]

The availability of the system using this approach increases when we scrub faster. Scrubbing faster means that we change the modules in the RP4 faster and therefore the probability of catching an error is higher than scrubbing at a lower rate.

Area-driven technique

The area-driven technique takes into account area information in order to check for time that is proportional to the area of the modules of the design. The metrics for the reconfigurable TMR design with the area-driven method will not be the same as the round robin technique. The number of essential bits of the RP4 is a weighted average of the essential bits of the modules of the design. Thus, we assume that we have a set of weights \( w_i, w_{i+1}, w_{i+2} \ldots w_n \) that they should have the following property \( \sum_{i=1}^{n} w_i = 1 \). Every weight is calculated as

\[
w_i = \frac{EB_i}{EB_i + EB_{RP1} + EB_{RP2} + EB_{Voter}}
\]

The total number of essential bits for the reconfigurable TMR design with the area-driven method is:

\[
EB (area-driven RP4) = \sum_{i=1}^{n} w_i \times EB_i
\]

\[
EB (Round robin) = EB (Round robin RP1) + EB (Round robin RP2) +

EB (Round robin Voter) + \sum_{i=1}^{n} w_i \times EB_i \Rightarrow
\]

120
Therefore the SEU rate for the area-driven technique is calculated below:

\[
    f_{seu} \text{ (area-driven)} = f_{seu} \text{ (baseline)} \times \frac{EB \text{ (area-driven)}}{EB \text{ (baseline)}}
\]

Having a different \( f_{seu} \) rate than the round robin requires different scrubbing rate \( f_{scrub} \) for changing the modules in the RP2 region in order to keep the same the ratio \( \frac{f_{scrub}}{f_{seu}} \) and make the comparisons with the other designs.

The \( MTTR \) of the area-driven method is calculated as shown below:

\[
    MTTR \text{ (area-driven)} = MTTR \text{ (area-driven RP1)} + MTTR \text{ (area-driven RP2)} + \\
    MTTR \text{ (area-driven Voter)} + \sum_{i=1}^{n} w_i \times MTTR_i
\]

The availability of the reconfigurable TMR with the area-driven technique will be different than the availability of the round robin scheduling due to the change in the \( MTTR \) and the \( f_{seu} \) and \( f_{scrub} \). Therefore the downtime will be different, and the prediction is that the results will be better as more faults occur in the large area modules and checking them more time we will detect errors faster.

### 5.11 TMR version of the second order polynomial

The commonly used technique for masking errors is TMR. A common way of implementing a TMR design on FPGAs is to place the three replicas in separate reconfigurable partitions and the voter logic in another one. Figure 5.14 shows the block diagram of the polynomial TMR system. The block diagram consists of three replicas of the second order polynomial and a voter logic. The voter logic receives three inputs, which are the outputs of the three replicas and produces the output of the IP plus three error signals. The logic inside the voter circuit includes a majority voter that consists of three AND gates and one OR gate. The output of the OR gate drives three XOR gates in order to detect which of the three replicas is in error condition.

The three error signals are connected to three interrupts that are being monitored
by the ARM CPU. Whenever any one of the three is asserted, the respective replica is scrubbed together with the voter module.
Figure 5.14: Polynomial TMR
5.12 Proposed SEU Mitigation Technique for a partial TMR system

5.12.1 High level view

A novel architecture able to partially mask errors at the application level due to SEUs is introduced in this section providing an alternative solution to TMR by trading-off availability for resources.

The main idea of the new scheme is to cyclically load operators of the main datapath to a reconfigurable region and form a partial TMR scheme. Basically, two replicas are operated in DMR mode and there is an extra region that can be dynamically reconfigured with any one of the modules of a replica, therefore forming in essence a partially TMR system.

5.12.2 Low level view

Figure 5.15 illustrates the application of the above scheme on the second-order polynomial datapath. The proposed IP consists of four reconfigurable partitions. Two of them are identical and contain the full datapath with some extra multiplexers that get as an input either the output of the previous stage or the voter output that comes from the voter logic partition. Three control registers set the multiplexers appropriately based on the current module that is in TMR mode.

The voter logic partition includes four multiplexers, three AND gates and one OR gate, six XOR gates and a control register. Multiplexer mux9 selects one of the inputs of the operators of replica 1 and multiplexer mux10 selects one of the inputs of the operators of replica 2. The outputs of the two multiplexers are compared with an XOR gate, and in case of a mismatch, the reconf input1 error will be set. This signal is connected to an interrupt to inform the processor to enable a full scrub of the four partitions. In addition, the output of the mux9 drives the first input of the operator that is loaded on the reconfigurable replica. It should be noted that a variable in C code indicates which
operator is currently loaded in the reconfigurable replica, in order to load a fresh copy of
the same module in the case of the aforementioned interrupt event.

The other two multiplexers, mux11 and mux12 receive as inputs, the outputs of
the operators of each replica respectively. Thus, mux11 is driven by the outputs of the
operators of replica 1 and mux12 selects one of the four outputs of the operators of replica
2. Furthermore, the outputs of the last two multiplexers together with the output of the
reconfigurable replica are connected to a majority voter logic that is formed by the three
AND gates and the OR gate. The majority voter output is connected to the first input
of every multiplexer of replica 1 and replica 2 for enabling masking at the corresponding
stage, and not allowing a single error propagating all the way to the output of a replica.
By having this hardware in place, TMR can be provided for one operator of the single
datapath at a time.

The majority voter output is also used for comparing with the two outputs of mux11
and mux12, in order to identify which replica is in error condition. The three XOR gates
provide error signals for replica 1, replica 2 and reconfigurable replica. The three error
signals are driving three interrupts to inform the ARM CPU which replica to scrub. In
this way, area and power can be saved, as there is no need to scrub all the four partitions.

Finally, one XOR gate is used to compare the register 1 and register 2 outputs. The
last XOR gate is comparing the output a12 of the last adder of replica 1 with output a22
that is the output of the last adder of replica 2. Whenever the DMR error signal of this
XOR gate is asserted, the respective interrupt is activated and a full scrub of the four
partitions is performed in order to correct the error.
Figure 5.15: Reconfigurable TMR Polynomial
5.13 Performance Evaluation of FPGA scrubbing techniques (TMR)

The TMR and the Reconfigurable TMR designs have been evaluated in exactly the same way with the DMR and the Reconfigurable DMR designs in terms of availability, MTTR, size of data transferred from the DDR and area.

5.13.1 SEU rate

In order to calculate the SEU and scrubbing rate, the number of essential bits of design should be taken into account. Table 5.7 illustrates the breakdown of essential bits for each of the partitions of the TMR and the Reconfigurable TMR designs. The assumption made in the case of the DMR study for the SEU rate and the number of essential bits stands here as well.

Table 5.7: SEU rates and size of essential bits for the on-demand scrubbing (TMR) and round robin scheduling and area-driven scrubbing techniques (Reconfigurable TMR)

<table>
<thead>
<tr>
<th>Partitions</th>
<th>Number of essential bits</th>
<th>SEU rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RP1</td>
<td>78,683</td>
<td></td>
</tr>
<tr>
<td>RP2</td>
<td>78,683</td>
<td></td>
</tr>
<tr>
<td>RP3</td>
<td>78,683</td>
<td>3.08</td>
</tr>
<tr>
<td>Voter</td>
<td>6,452</td>
<td></td>
</tr>
<tr>
<td>Reconfigurable TMR (round robin scheduling)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RP 1</td>
<td>87,032</td>
<td></td>
</tr>
<tr>
<td>RP 2</td>
<td>87,032</td>
<td></td>
</tr>
<tr>
<td>RP 3</td>
<td>10,310 (average)</td>
<td>2.57</td>
</tr>
<tr>
<td>Voter</td>
<td>18,090</td>
<td></td>
</tr>
<tr>
<td>Reconfigurable TMR (area-driven)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RP 1</td>
<td>87,032</td>
<td></td>
</tr>
<tr>
<td>RP 2</td>
<td>87,032</td>
<td></td>
</tr>
<tr>
<td>RP 3</td>
<td>15,395 (weighted average)</td>
<td>2.63</td>
</tr>
<tr>
<td>Voter</td>
<td>18,090</td>
<td></td>
</tr>
</tbody>
</table>

The number of essential bits of the TMR design is the sum of the number of essential bits of RP 1, RP 2, RP 3 and voter and equals 242,501. The SEU rate of the TMR design is computed in Equation 5.19.
\[ SEU \text{ rate of the TMR} = \frac{\# \text{ essential bits of the TMR}}{\# \text{ essential bits of the baseline}} \]

\[ \times SEU \text{ rate of the Baseline} \]
\[ \frac{242,501}{78,666} = 3.08 \] (5.19)

The SEU rate of the Round robin scheduling for the reconfigurable TMR is computed in Equation 5.25 by first calculating the average number of essential bits of the RP3 partition which is the same with Equation 5.5.

\[ SEU \text{ rate of the round robin scheduling} = \frac{RP1 + RP2 + RP3 + voter}{\# \text{ essential bits of the baseline}} \]

\[ \times SEU \text{ rate of the Baseline} \]
\[ = 2.57 \] (5.20)

For the area-driven scrubbing technique, a weighted average of the number of essential bits of the RP3 partition is computed. The weights are computed based on the percentage of essential bits of each operator over the total number of essential bits, when the same operator is placed in the RP3 partition, as shown in Equation 5.21. It should be mentioned that the weight of ADD operator and the weight of MULT1 operator are equal since they both include the same number of essential bits. The number of essential bits of the ADD, MULT1 and MULT2 have been calculated in Table 5.2.

\[ Weight1 = \frac{ADD}{RP1 + RP2 + ADD + Voter} = 0.03 \] (5.21)

Equation 5.22 computes the weight of the two-input multiplier, that is the largest module in terms of area of the polynomial IP.
\[
\text{Weight}_2 = \frac{\text{MULT}_2}{\text{RP}_1 + \text{RP}_2 + \text{ADD} + \text{Voter}} = 0.11
\] (5.22)

The two weights should sum up to 1, since this would be an entire cycle through the operators. However, the addition of weight 1 (multiplied by 3 for the two adders and the constant coefficient multiplier) and weight 2 equals 0.2. Therefore, the sum of the two weights is scaled to 1 as shown in Equation 5.9.

\[
\text{NewWeight}_1 = \frac{0.03 \times 100 \%}{0.2} = 15\
\] (5.23)

\[
\text{NewWeight}_2 = \frac{0.22 \times 100 \%}{0.2} = 55\
\]

Thus, according to Equation 5.23, the weighted average of the essential bits of the RP3 for the area-driven technique is shown in Equation 5.24

\[
\text{RP}_3 = 2 \times 0.15 \times \text{ADD} + 0.15 \times \text{MULT}_1 + 0.55 \times \text{MULT}_2 = 15,395
\] (5.24)

Finally, Equation ?? computes the SEU rate of the area-driven scrubbing technique for the reconfigurable DMR polynomial.

\[
\text{SEU rate of area-driven} = \frac{\text{RP}_1 + \text{RP}_2 + \text{RP}_3 + \text{voter}}{\# \text{ essential bits of the baseline}} \\
* \text{SEU rate of the Baseline}
\] (5.25)

\[
= 2.63
\]

The ratio fscrub/fseu should be kept the same for the TMR and the reconfigurable TMR designs. Table 5.8 indicates the scrubbing rates and SEU rates for the round robin scheduling and area-driven scrubbing techniques.
Table 5.8: fscrub/fseu, SEU and scrubbing rates for round robin scheduling and area-driven techniques of the Reconfigurable TMR design

<table>
<thead>
<tr>
<th></th>
<th>fscrub/fseu</th>
<th>fscrub(event/sec)</th>
<th>fseu (event/sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Round robin scheduling</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/16</td>
<td>0.160625</td>
<td>2.57</td>
<td></td>
</tr>
<tr>
<td>1/8</td>
<td>0.32125</td>
<td>2.57</td>
<td></td>
</tr>
<tr>
<td>1/4</td>
<td>0.6425</td>
<td>2.57</td>
<td></td>
</tr>
<tr>
<td>1/2</td>
<td>1.285</td>
<td>2.57</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2.57</td>
<td>2.57</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>5.14</td>
<td>2.57</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>10.28</td>
<td>2.57</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>20.56</td>
<td>2.57</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>41.12</td>
<td>2.57</td>
<td></td>
</tr>
<tr>
<td><strong>Area-driven</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/16</td>
<td>0.164375</td>
<td>2.63</td>
<td></td>
</tr>
<tr>
<td>1/8</td>
<td>0.32875</td>
<td>2.63</td>
<td></td>
</tr>
<tr>
<td>1/4</td>
<td>0.6575</td>
<td>2.63</td>
<td></td>
</tr>
<tr>
<td>1/2</td>
<td>1.315</td>
<td>2.63</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2.63</td>
<td>2.63</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>5.26</td>
<td>2.63</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>10.52</td>
<td>2.63</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>21.04</td>
<td>2.63</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>42.08</td>
<td>2.63</td>
<td></td>
</tr>
</tbody>
</table>

5.13.2 Availability

Figure 5.16 illustrates the availability of all different techniques that have been investigated in this chapter for several ratios of scrub over SEU rate. The y-axis includes the confidence intervals of the availability for the experiments that were run. For the reconfigurable TMR case, the Round robin scheduling and the area-driven scrubbing techniques do not differ that much as the scrubbing rate is increased. This is due to the fact that this particular design has DMR in place plus partial TMR for its operators. Therefore, with the DMR in place, the availability achieved is very high.
Figure 5.16: Availability for scrubbing techniques
5.13.3 Area resources

Table 5.9 represents area results for the TMR and the Reconfigurable TMR designs when implemented on the ZYNQ platform. The TMR occupies 1571 LUTs and 346 REGs while the reconfigurable TMR saves 15% LUTs and 23% REGs over TMR when the round robin scheduling technique is applied and 12% LUTs and 23% REGs when the area-driven method is used.

More savings in resources could be achieved if another case study that occupies more area will be used because the overheads of the techniques being used will be smaller as compared to the actual modules of the case study. In the case study of this work, the area overheads for implementing the scrubbing techniques are comparable in size with the modules that are protected, i.e. adder, multiplier.
Table 5.9: Resources utilisation of the TMR and Reconfigurable TMR designs

<table>
<thead>
<tr>
<th></th>
<th>RPs</th>
<th>#LUTs</th>
<th># REGs</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RP1</td>
<td></td>
<td>508 out of 1600 (31.75%)</td>
<td>114 out of 3200 (3.56%)</td>
</tr>
<tr>
<td>RP2</td>
<td></td>
<td>508 out of 1600 (31.75%)</td>
<td>114 out of 3200 (3.56%)</td>
</tr>
<tr>
<td>RP3</td>
<td></td>
<td>508 out of 1600 (31.75%)</td>
<td>114 out of 3200 (3.56%)</td>
</tr>
<tr>
<td>Voter</td>
<td></td>
<td>47 out of 1600 (2.94%)</td>
<td>4 out of 3200 (0.1%)</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>1,571 out of 1600 (98.18%)</td>
<td>346 out of 3200 (10.81%)</td>
</tr>
<tr>
<td>Reconf. TMR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RP1</td>
<td></td>
<td>570 out of 1600 (35.63%)</td>
<td>118 out of 3200 (3.69%)</td>
</tr>
<tr>
<td>RP2</td>
<td></td>
<td>570 out of 1600 (35.63%)</td>
<td>118 out of 3200 (3.69%)</td>
</tr>
<tr>
<td>RP3(ADD)</td>
<td></td>
<td>32 out of 1600 (2%)</td>
<td>16 out of 3200 (0.5%)</td>
</tr>
<tr>
<td>RP3(MULT1)</td>
<td></td>
<td>32 out of 1600 (2%)</td>
<td>16 out of 3200 (0.5%)</td>
</tr>
<tr>
<td>RP3(MULT2)</td>
<td></td>
<td>166 out of 1600 (10.38%)</td>
<td>16 out of 3200 (0.5%)</td>
</tr>
<tr>
<td>Voter</td>
<td></td>
<td>129 out of 1600 (8.06%)</td>
<td>14 out of 3200 (0.4%)</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>1335 out of 1600 (83.4%)</td>
<td>266 out of 3200 (8.31%)</td>
</tr>
<tr>
<td></td>
<td>(round robin scheduling)</td>
<td>15% saving over TMR</td>
<td>23% saving over TMR</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>1,375 out of 1600 (86%)</td>
<td>266 out of 3200 (8.31%)</td>
</tr>
<tr>
<td></td>
<td>(area-driven)</td>
<td>12.5% saving over TMR</td>
<td>23% saving over TMR</td>
</tr>
</tbody>
</table>
5.13.4 Mean Time to repair (MTTR)

The resources utilisation should be taken into account in order to calculate the MTTR of each design. The TMR system consists of four reconfigurable partitions, the three of them being identical in area resources plus the voter that occupies one partition by itself. Equations 5.26, 5.27, 5.28 compute the MTTR of the TMR design. It should be noted that the MTTR of the TMR design considers only one of the (RP1,RP2,RP3) plus the MTTR of the voter, since only the faulty replica and the voter replica are scrubbed.

\[
\begin{align*}
MTTR_{RP1} &= MTTR_{RP2} = MTTR_{RP3} = \\
&= 7 \text{ msec} \times (\% \text{ RP1 LUTs} + \% \text{ RP1 REGs}) = 2.47 \text{ msec}
\end{align*}
\]  

\[
MTTR_{Voter} = 7 \text{ msec} \times (\% \text{ Voter LUTs} + \% \text{ Voter REGs}) = 0.21 \text{ msec}
\]  

\[
MTTR_{TMR} = MTTR_{RP1} + MTTR_{Voter} = 2.68 \text{ msec}
\]  

The Reconfigurable TMR consists of two identical in size RPs, another partition (RP3) that accommodates one of the four operators of the polynomial and the voter occupying a fourth partition. One MTTR is calculated for the Round robin scheduling case and another one for the area-driven technique.

The MTTR of the RP1 is identical for the round robin scheduling and the area-driven techniques and calculated as shown in Equation 5.29.

\[
MTTR_{RP1} = MTTR_{RP2} = 7 \text{ msec} \times (\% \text{ RP1 LUTs} + \% \text{ RP1 REGs}) = 2.75 \text{ msec}
\]  

The MTTR of the ADD, MULT1 and MULT2 are not calculated again, because they are the same with the reconfigurable DMR design. However, the MTTR of the voter is measured using Equation 5.30.
\[ \text{MTTR}_{\text{Voter}} = 7 \text{ msec} \times (\% \text{ Voter LUTs} + \% \text{ Voter REGs}) = 0.59 \text{ msec} \quad (5.30) \]

The MTTR of the Round robin scheduling technique is calculated based on Equations 5.15, 5.16 and is presented in Equation 5.31.

\[
\text{MTTR}_{\text{round robin scheduling}} = 2 \times \text{MTTR}_{\text{RP1}} + \text{MTTR}_{\text{Voter}} + \frac{3 \times \text{MTTR}_{\text{ADD}} + \text{MTTR}_{\text{MULT2}}}{4} \quad (5.31)
\]

\[= 6.41 \text{ msec} \]

The MTTR of the area-driven technique is calculated by the Equation 5.32 based on the weights that have been calculated earlier in Equation 5.23.

\[
\text{MTTR}_{\text{area-driven}} = 2 \times \text{MTTR}_{\text{RP1}} + 3 \times 0.15 \times \text{MTTR}_{\text{ADD}}
+ 0.55 \times \text{MTTR}_{\text{MULT2}}
+ \text{MTTR}_{\text{Voter}} \quad (5.32)
\]

\[= 6.58 \text{ msec} \]

Table 5.10 illustrates the MTTR for TMR and Reconfigurable TMR designs. It can be concluded that round robin scheduling has 15.9% smaller MTTR as compared to the TMR while the area-driven scrubbing method achieves 13.65% lower MTTR than TMR.

<table>
<thead>
<tr>
<th>Table 5.10: MTTR for TMR and Reconfigurable TMR designs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MTTR in msec</strong></td>
</tr>
<tr>
<td>TMR</td>
</tr>
<tr>
<td>Reconfigurable TMR (round robin scheduling)</td>
</tr>
<tr>
<td>Reconfigurable TMR (area-driven)</td>
</tr>
</tbody>
</table>
5.13.5 Size of data transferred from DDR3 for scrubbing

Table 5.11 includes the bitstream size for each one of the two designs. The bitstream size of the TMR is the 64,117 bytes. The Reconfigurable TMR design has 23,130 bytes for RP1 and RP2, 5,002 bytes for the voter and 1,471 or 6,402 bytes for the RP3 partition depending on the operator that is placed on it. It should be noted that the bitstream size of RP1 of the Reconfigurable TMR is larger than the RP1 of the TMR design, due to the extra resources for implementing the partial TMR masking for the case of the Reconfigurable TMR.

Table 5.11: Bitstream size in bytes for the three designs

<table>
<thead>
<tr>
<th>RPs</th>
<th>Bitstream size in bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMR</td>
<td></td>
</tr>
<tr>
<td>RP1</td>
<td>20,776</td>
</tr>
<tr>
<td>RP2</td>
<td>20,776</td>
</tr>
<tr>
<td>RP3</td>
<td>20,776</td>
</tr>
<tr>
<td>Voter</td>
<td>1,789</td>
</tr>
<tr>
<td>Reconf. TMR</td>
<td></td>
</tr>
<tr>
<td>RP1</td>
<td>23,130</td>
</tr>
<tr>
<td>RP2</td>
<td>23,130</td>
</tr>
<tr>
<td>RP3(ADD)</td>
<td>1,471</td>
</tr>
<tr>
<td>RP3(MULT1)</td>
<td>1,471</td>
</tr>
<tr>
<td>RP3(MULT2)</td>
<td>6,402</td>
</tr>
<tr>
<td>Voter</td>
<td>5002</td>
</tr>
</tbody>
</table>

Figure 5.17 demonstrates the trade-off between unavailability and size of data transferred from DDR3 per technique, having the origin of the diagram as the best performing point. The best design that performs better on this trade-off is the TMR design that implements the on-demand scrubbing technique. It achieves very high availability with small power consumption for scrubbing, due to the few memory transfers from the DDR. The few memory transfers for scrubbing exist because we have partitioned the TMR design in three reconfigurable partitions and in the case of a fault in one of its replicas, we scrub only the affected replica and the voter, and not the entire TMR design. The area-driven approach for the Reconfigurable TMR design performs better than the round robin scheduling due to the area information extracted from the modules that are par-
ially masked. Thus, larger modules in the datapath are protected for more time according to this technique, whereas the round robin scheduling uses the same amount of time for checking for errors for every module. The large amount of memory transfers of the Reconfigurable TMR design is due to the fact that the second-order polynomial case study is a very small design and the overheads for implementing the scrubbing techniques are similar with the size of the modules. Therefore, a larger in size case study, would move the data points correspond to the round robin scheduling and area-driven methods for Reconfigurable TMR to the right closer to the origin of the graph.

The round robin scheduling and area-driven techniques for the Reconfigurable TMR design achieve much higher availability than DMR but at a higher cost of DDR3 data transfers for scrubbing.

Figure 5.17: Unavailability vs Total size of data transferred from DDR3 memory
5.14 Validation of the formal model (TMR)

Figure 5.18 shows the comparison of the availability results for the second order polynomial case study taken from the formal model and the real environment experiments. The focus is on three techniques: a) the round robin scheduling for the reconfigurable TMR design, b) the area-driven technique for the reconfigurable TMR design and c) the on-demand scrubbing for the TMR. For each technique, the availability results from the formal model and the empirical data are side by side for different ratios of $\frac{f_{\text{scrub}}}{f_{\text{seu}}}$. The points on the x-axis (1 ... 19) with interval 1 are taken from the formal model, while points on the x-axis (1.5 ... 19.5) represent the results from the experiments. The availability results from the formal model slightly differ from the empirical results due to the assumptions made when developing the theoretical model.

The SEUs are drawn from an exponential distribution with the same rate for both the formal model and the experimental results. We assume that only 1% of the SEUs are causing an error in the TMR design and this is because TMR can mask a single error and only errors in the voter will propagate to the output of the design. For the reconfigurable TMR design, the assumption was that 5% of the SEUs will produce an error that will be detected in different time intervals for the round-robin and the area-driven techniques.
Figure 5.18: Comparison of the formal model with the actual data (TMR)
5.15 Conclusion

In conclusion, two new scrubbing techniques taking into account area information and using partial reconfiguration were presented in this chapter. A comparison with existing scrubbing techniques shows that the first area-driven technique for the Reconfigurable DMR design saves 43.6% LUTs and 40.9% REGs when compared to an on-demand scrubbing strategy based on DMR. The design space exploration indicates that some designs of the proposed technique perform better than blind scrubbing on the availability versus data transferred from DDR3 memory trade-off for the second order polynomial IP and therefore, power savings are achieved as compared to the baseline design with blind scrubbing. The on-demand scrubbing technique achieves very high availability but at the cost of very large resource overhead as compared to the baseline and the Reconfigurable DMR.

The second area-driven scrubbing technique for the reconfigurable TMR design saves 15% LUTs and 23% REGs when compared to the on-demand scrubbing strategy based on TMR but it does not achieve the same availability as the TMR. However, a larger case study would show significant savings in the data transferred from DDR and therefore, power savings as the one used in this work due to the fact that the overheads of the scrubbing techniques would be smaller than the actual size of the modules of the case study.

Finally, the experimental results show that area savings can be achieved using partial reconfiguration techniques that trade-off availability for resources. Moreover, power savings as compared to DMR and TMR can be achieved using dynamic partial reconfiguration, by transferring less data from external memory for scrubbing.
Chapter 6

Conclusion

6.1 Current Status

Aggressive transistor scaling combined with ageing, degradation and radiation create high reliability risks for integrated circuits. The prediction for even higher failure rates for logic and memory cells in the future was the motivation behind this work. The literature for reliability and fault-tolerance in FPGAs is rich, but novel architectures and techniques were developed in this thesis that reduce the probability of failure and increase the lifetime of applications such as an artificial pancreas, and provide higher dependability at reduced costs.

More specifically, a hybrid-substrate has been proposed in this thesis for providing fault-tolerance capabilities to a safety-critical application, namely an artificial pancreas design. The proposed architecture is able to detect and correct transient and permanent faults, utilising DMR on a per-module basis along with ASIC and FPGA-like reconfiguration to correct permanent faults. The implementation results show that in terms of hardware dependability a total safe failure fraction $SFF_{total}$ of 99% and total probability of failure per hour $PFH_{total}$ that equals $2.32 \times 10^{-12}$ can be achieved. The value of $SFF_{total}$ means that the required SIL 3 level is achieved for the particular application. In comparison to existing fault-tolerant techniques, the proposed architecture achieves 5,100x lower PFH than a DMR for permanent faults with 2.4x the area of the DMR. In addi-
tion, the proposed solution achieves 83x lower PFH than a TMR with 1.6x area overheads when considering transient faults. The reported overhead in area is due to the FPGA-like fabric that was used in the proposed architecture. The current area ratio between FPGA-like substrate and ASIC is 5 and if this ratio decreases in the future the area and power overheads of the proposed architecture will be further reduced.

The main goal of the proposed technique here was to extend the life time of the artificial pancreas using an FPGA-like substrate that offers the flexibility for on-demand system reliability. The limitation of the technique is the area overheads introduced by this custom fabric. Apart from the specific application the technique can be used to other applications but is not suitable for high performance applications because of the performance degradation due to the FPGA-like fabric and due to the fact that the system need to be taken off-line in order to be tested and identify errors. In designs that are partitioned into modules such that the sum of their area minus the size of the larger module is greater in size than the total area of the multiplexers and the area ratio \( \frac{FPGA}{ASIC} = 1 \), our technique will reduce the probability of failure while also offering area savings when compared to a TMR design. Moreover, the more modules we partition a design, the larger the multiplexers will need to be designed leading to extra area overheads.

A new framework for experimenting with multiple scrubbing techniques was implemented on a modern FPGA device, the ZYNQ. Fault injection is achieved using the Xilinx SEM core and FPGA scrubbing is supported using partial reconfiguration via the ICAP primitive. Any type of distribution for the inter-arrival time of particles is supported by the framework, and it consists a testing platform for engineers and researchers for applying different scrubbing strategies and measuring several dependability metrics. Experimental results show that it is a low area design occupying only 4.2% of the LUTs, 8.6% of the registers and 65% of the BRAMs on the Zedboard. The framework supports various SEU and scrub rates, but it is limited to the value of \( f_{\text{scrub}}/f_{\text{seu}} = 16 \), due to a mean error of \( 4.69 \times 10^{-4} \) and standard deviation 1.4e-3 between the targeted (theoretical) and the actual fault injections in time. This is due to the increased number of scrubs placed in time between SEU events, leading to a time shift of the SEUs and therefore, deviating from
the targeted distribution. Effectively there is a trade-off between SEU rate, fscrub/fseu ratio and how long the experiment lasts. Therefore, selecting a lower SEU rate, increasing the scrubbing rate and running the experiments longer would decrease the values of errors between the original and the actual SEU distributions. The framework was used to evaluate different scrubbing strategies applied on various versions of designs (baseline, DMR, TMR).

Beside the implementation of existing scrubbing techniques such as blind scrubbing and on-demand scrubbing, a novel technique that is driven by taking into account area information of the application was proposed in chapter 5. Experimental results show that our technique saves 43.6% LUTs and 40.9% REGs when compared to an on-demand scrubbing strategy based on DMR for a second order polynomial IP. Finally, the design space exploration indicates that some designs of the proposed area-driven technique perform better than blind scrubbing on the availability versus power consumption trade-off for the targeted application. An alternative technique using a partial TMR architecture and area-driven scrubbing was investigated against the full TMR version of the second-order polynomial IP. The results show that our technique saves 15% LUTs and 23% REGs as compared to TMR but it spends more power consumption and achieves almost the same availability with the TMR.

A formal model was developed and the availability results taken from the model were compared to experimental results. The observation was that for the blind scrubbing and the on-demand scrubbing strategies the model converges to the experimental results. However, the round robin and area-driven techniques were more difficult to model, since the detection of an error differs in theory and in practice. However, this model can be used to estimate changes in expected properties such as area overheads and availability of more complex designs.

The scrubbing techniques developed in this thesis can be applied in other applications and the prediction is that these techniques will offer more clear benefits when bigger case studies will be under investigation. The aforementioned techniques suit better feed-forward and state-less datapaths such as FIR filters and other filters from the DSP domain. The
proposed techniques can be enriched with check-pointing and rollback functions in order to be used in hardware designs that include feedback paths and complex control logic.

The final conclusion of this thesis is that partial reconfiguration techniques on FPGA-based systems can increase the reliability of a safety critical application with increased area overheads that can be reduced in the future if custom FPGA-like substrates are used. Furthermore, partial reconfiguration can be used for applications that there they do not require 100% availability such as image processing, telecommunications and networks in space air crafts for reducing the area and power overheads of DMR and TMR.

6.2 Future Work

A promising research direction to continue the work related to the artificial pancreas is to investigate a new partition algorithm that allows grouping of the modules in sets of two or more as per partition and evaluate their failure rate. In addition, the techniques developed in chapter 3 can be applied in a range of applications such as an FIR filter and a soft processor. Furthermore, a high level model of the overheads of the fault-tolerant techniques could be devised to help estimate a priori the suitability of the technique to various applications. An attempt to develop a custom FPGA-like fabric with reduced functionality and resources would be a potential solution to decrease area and power overheads as compared to the Xilinx FPGA fabric. Other fault detection techniques such as CED or reduced precision redundancy could be applied to the multiplier and the adder modules of the artificial pancreas application and investigate the trade-off between area and tolerance to permanent and transient faults.

The novel scrubbing techniques developed in Chapter 5 could be applied to other case studies such as an FIR filter and a soft processor. Moreover, an investigation of the effects of multi-bit upsets can be made to analyse the robustness of the second-order polynomial and the other case studies to multiple SEUs. The framework can be extended to be fully automated and parametrisable to help researchers and practitioners easily implement their applications. Moreover, a decrease in resources would be the result of introducing DMR and TMR only on the MSBs of the datapath instead of applying modular redundancy and
the investigation of the trade-off availability versus power would be an interesting research result. Finally, instead of taking into account area information, a sensitivity analysis on the reliability of the modules of the case study can be made based on which the time each module is checked, depends on their sensitivity to SEUs. For example, if the multiplier has higher sensitivity to SEUs, will be checked for more time as compare to the rest of the modules.
Bibliography


On-demand system reliability, Microprocessors and Microsystems, Volume 37, Issue 8, Part C, November 2013, Pages 981-1001, ISSN 0141-9331.


[35] Xilinx TMRtool, First Triple Modular Redundancy Development tool for reconfigurable FPGAs

[36] Brigham Young University. 2014. BYU EDIF Tools Home Page

[37] Mentor Graphics. 2014. Precision Hi-Rel Technology Overview

[38] Synplify Premier. Fast, Reliable FPGA Implementation and Debug


[72] M. Berg et al., ”Effectiveness of internal vs. external SEU scrubbing mitigation strategies in a Xilinx FPGA: Design, test, and analysis,” Radiation and Its Effects on Com-


[77] IEC/TR 62380:2004(E), Reliability data handbook Universal model for reliability prediction of electronics components, PCBs and equipment


[84] Safety Integrity Level Manual, Pepperl and Fuchs


[86] IEC International standard 60601 for medical electrical equipment.


[91] Soft Error Mitigation using Prioritised Essential Bits.XAPP 538 v.1.0

[92] Xilinx Soft Error Mitigation Controller v4.1 Product Guide


155