FAULT DETECTION, ISOLATION AND RECOVERY SCHEMES FOR SPACEBORNE RECONFIGURABLE FPGA-BASED SYSTEMS

PhD Thesis

FELIX SIEGLE

University *of* Leicester

Department of Engineering
University of Leicester

Supervisor: Tanya Vladimirova

October 2015

Felix Siegle: *Fault Detection, Isolation and Recovery Schemes for Space-borne Reconfigurable FPGA-Based Systems*, PhD Thesis, © October 2015
LOCATION:
Leicester, United Kingdom / Noordwijk, The Netherlands

ABSTRACT

This research contributes to a better understanding of how reconfigurable Field Programmable Gate Array (FPGA) devices can safely be used as part of satellite payload data processing systems that are exposed to the harsh radiation environment in space. Despite a growing number of publications about low-level mitigation techniques, only few studies are concerned with high-level Fault Detection, Isolation and Recovery (FDIR) methods, which are applied to FPGAs in a similar way as they are applied to other systems on board spacecraft.

This PhD thesis contains several original contributions to knowledge in this field. First, a novel *Distributed Failure Detection* method is proposed, which applies FDIR techniques to multi-FPGA systems by shifting failure detection mechanisms to a higher intercommunication network level. By doing so, the proposed approach scales better than other approaches with larger and complex systems since data processing hardware blocks, to which FDIR is applied, can easily be distributed over the intercommunication network. Secondly, an innovative *Availability Analysis* method is proposed that allows a comparison of these FDIR techniques in terms of their reliability performance. Furthermore, it can be used to predict the reliability of a specific hardware block in a particular radiation environment. Finally, the proposed methods were implemented as part of a proof of concept system: On the one hand, this system enabled a fair comparison of different FDIR configurations in terms of power, area and performance overhead. On the other hand, the proposed methods were all successfully validated by conducting an accelerated proton irradiation test campaign, in which parts of this system were exposed to the proton beam while the proof of concept application was actively running.

Manche Menschen benutzen ihre Intelligenz
zum Vereinfachen, manche zum Komplizieren.

— Erich Kästner

ACKNOWLEDGMENTS

Sponsorship from ESA under the NPI Programme, Airbus Defence and Space, UK and the University of Leicester is gratefully acknowledged.

Many thanks to my supervisors: Tanya Vladimirova from University of Leicester, Omar Emam from Airbus Defence and Space, UK and Jørgen Ilstad, Martin Suess and Christian Poivey from ESA.

I would also like to thank my family, my friends and my colleagues at ESA-ESTEC.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

ABFT Algorithm Based Fault Tolerance

AMBA Advanced Microcontroller Bus Architecture

API Application Programming Interface

ASIC Application-Specific Integrated Circuit

BRAM Block RAM

CCSDS Consultative Committee for Space Data Systems

CLB Configurable Logic Block

COTS Commercial Off-The-Shelf

CPU Central Processing Unit

CRC Cyclic Redundancy Check

CNES Centre National d'Études Spatiales

DCE Domain Crossing Error

DCM Digital Clock Manager

DMA Direct Memory Access

DFS Depth-First Search

DRPM Dynamically Reconfigurable Processing Module

DSP Digital Signal Processing

DUT Device Under Test

DWC Duplication with Compare

ECC Error-Correcting Code

ECSS European Cooperation for Space Standardization

EDAC Error Detection and Correction

EEP Error End of Packet

EOP End of Packet

ESA European Space Agency

FAR Frame Address Register

FIFO First In, First Out

FCT Flow Control Token

FDIR Fault Detection, Isolation and Recovery

FMEA Failure Mode and Effects Analysis

FPGA Field Programmable Gate Array

FPS Frames Per Second

GEO Geostationary Earth Orbit

GPIO General Purpose Input Output

HDL Hardware Description Language

IC Integrated Circuit

ICAP Internal Configuration Access Port

IOB Input Output Block

IP Intellectual Property

JPEG Joint Photographic Experts Group

JPL Jet Propulsion Laboratory

LEO Low Earth Orbit

LET Linear Energy Transfer

LUT Look-Up Table

MAC Multiply and Accumulate

MBU Multiple Bit Upset

MEO Medium Earth Orbit

MTBF Mean Time Between Failure

MTTR Mean Time to Recover

NASA National Aeronautics and Space Administration

NoC Network on Chip

OPB On-chip Peripheral Bus

PCB Printed Circuit Board

PIP Programmable Interconnect Point

POR Power-On-Reset

POSIX Portable Operating System Interface

PSI Paul Scherrer Institut

RGB Red, Green and Blue

RAM Random Access Memory

ROM Read Only Memory

RPR Reduced Precision Redundancy

SECDED Single Error Correction and Double Error Detection

SEE Single Event Effect

SEFI Single Event Functional Interrupt

SEL Single Event Latchup

SET Single Event Transient

SEU Single Event Upset

SMAP SelectMAP

SoC System on Chip

SoCP SoCWire Protocol

SPE Solar Particle Event

SRAM Static Random Access Memory

TID Total Ionising Dose

TMR Triple Modular Redundancy

UML Unified Modeling Language

1

CHAPTER 1: INTRODUCTION

1.1 WHY RECONFIGURABLE FPGAS IN SPACE?

Nowadays, Field Programmable Gate Arrays (FPGAs) are commonly used on board spacecraft. The importance of these devices for space applications is illustrated by the figures given in Table 1, which show that the vast majority of Integrated Circuits (ICs) on board the Sentinel-2 spacecraft, a current mission of the European Space Agency (ESA), are FPGAs. As evidenced by Table 1, while Application-Specific Integrated Circuits (ASICs) and microcontrollers still play an important role for platform applications, payload processing applications are mainly implemented with FPGAs.

Today, three main types of space qualified FPGA technologies are employed in commercial products. The most common technology is antifuse, which is used by one-time programmable FPGAs. One advantage of these devices is their natural tolerance against radiation effects because the hardware configuration is fixed. In principle, these devices can still suffer from Single Event Upsets (SEUs) in user logic and embedded Random Access Memory (RAM) cells. However, radiation tolerant versions are available, which offer hardened user flip-flops by design, e.g. the RTAX and RTSX devices by Microsemi.

The second technology is based on Static Random Access Memory (SRAM) memory, i.e. the configuration of the FPGA is stored in volatile memory cells. An obvious benefit of these devices is the possibility to reconfigure the hardware in later design or even mission stages. Furthermore, some of these devices like the newer Virtex-4 and Virtex-5 FPGAs by Xilinx offer high performance and a large

Table 1: FPGA usage example: Sentinel-2 mission [2]

IC Type	Quantity Platform	Quantity Payload
FPGA	**118**	**37**
Custom ASIC	72	6
Microcontrollers	23	0
Standard ASIC	10	0

amount of logic and embedded memory resources as well as dedicated Digital Signal Processing (DSP) blocks. In contrast to their antifuse counterparts, SRAM-based FPGAs are in principle more susceptible to SEUs because the hardware configuration can be altered by radiation effects (e.g. Virtex-4QV and earlier devices by Xilinx). FPGAs with radiation hardened configuration memory are also available, however, e.g. Virtex-5QV devices by Xilinx or ATF280 devices by Atmel.

Recently, flash memory based FPGAs are also being considered for use in space projects, e.g. the ProASIC3 device by Microsemi. Similar to SRAM-based FPGAs they can be reconfigured and offer good performance. The usage of such devices on long space missions is, however, problematic due to their rather low immunity to the Total Ionising Dose (TID) effect and Single Event Latchups (SELs) [4].

Internal studies at the Jet Propulsion Laboratory (JPL) estimate that the raw, uncompressed data captured from spectroscopy instruments on board recently proposed U.S. missions could reach 1 to 5 Terabytes per day [5]. It is therefore recommended to drastically reduce the data volume which must be stored on board and later transmitted to Earth by transforming the raw measurements of payload instruments into intermediate results performing on-board processing. In a technology assessment National Aeronautics and Space Administration (NASA) scientists also found that Xilinx FPGAs are best suited for such high performance tasks due to their flexibility and their embedded DSP blocks compared to single board computers and DSP processors. Apart from the increase in performance, SRAM-based FPGAs offer the capability of being reconfigured - a feature not to be underestimated for space projects. Pingree describes the typical problem of one-time programmable FPGAs in [6]. For one of the instruments on the NASA Juno spacecraft to Jupiter the engineers had to design and program the FPGA design two years before launch. Since the FPGA was one-time programmable, it could not be changed or improved without an high impact to the project cost and schedule. Furthermore, as the spacecraft travels for five years to Jupiter, instrument calibration activities may be required during that time, in which the FPGA design cannot be changed too. With SRAM-based FPGAs, however, hardware updates could easily be applied in later design stages or even in-flight.

1.2 SCOPE AND OBJECTIVES

As illustrated in the previous section, the usage of reconfigurable FPGAs in space is particularly interesting for applications that require high performance computing capabilities, e.g. for data compression, encryption, filtering and extraction. Therefore, the Fault Detection, Isolation and Recovery (FDIR) schemes proposed in this PhD thesis are mainly aimed at such high performance applications. Since the algorithms for such tasks can be quite complex, it might be necessary to split up the application into several hardware modules, which can then be distributed over several FPGAs. This possible segmentation must be taken into account too, i.e. the proposed FDIR schemes must also be applicable to reconfigurable multi-FPGA systems.

Nearly all recent publications are concerned with SRAM-based Xilinx Virtex-4QV and Virtex-5QV FPGAs, as they are currently the only fast SRAM-based FPGAs, which are available in space-qualified versions. In addition, there is some interest in also using commercial and defence-grade SRAM-based FPGAs on board spacecraft. On the one hand, these devices are much cheaper than their space-qualified counterparts and are therefore of interest for low-cost space missions. On the other hand, there is no indication at present that the latest SRAM-based FPGA devices, like Xilinx Virtex-7 or Altera Stratix-10, will ever be available in space-qualified, radiation-hardened versions. If extreme performance is required for a specific mission, however, the usage of commercial devices might be justifiable. Except for the Xilinx Virtex-5QV device and the low-performance Atmel ATF280 device, all current (and most likely many future) SRAM-based FPGAs suffer from radiation-induced failures due to Single Event Effects (SEEs) in their configuration memory and user memory elements. Therefore, the proposed FDIR schemes must be as technology-independent as possible to ensure that they remain valid for this broad range of available SRAM-based FPGA devices.

It quickly became clear that no recent work exists, which thoroughly surveys this rich research field. Therefore, a literature review was conducted, which on its own became an original contribution to knowledge in this field and which was later published in one of the leading computing survey journals [7]. The state of the art can coarsely be broken down into: (i) work on low-level radiation mitigation techniques that can be applied during run-time, (ii) work on fault emulation, avoidance and analysis techniques that can be ap-

plied during design-time and (iii) work on research and demonstrator platforms, implementing one or several of the techniques mentioned above.

The literature survey revealed two major research gaps:

1. Most publications are concerned with low-level and technology-dependent mitigation techniques but only few studies deal with high-level, more technology-independent FDIR methods, which are applied to FPGAs in a similar way as they are applied to other systems on board spacecraft.

2. Although some publications cover reliability and availability analysis methods respectively, they are mainly used to demonstrate the capabilities of a particular mitigation technique. However, when applying a FDIR method to a reconfigurable FPGA-based system, the ability to predict its overall effect on the system reliability is essential. The prediction must at least take into account the configuration memory and user RAM blocks of the SRAM-based FPGA since these elements are most susceptible to radiation effects. Then, the analysis method can be used (i) to predict the reliability of a certain hardware implementation in a specific radiation environment and (ii) to compare different FDIR approaches against each other.

To fill the first gap, a novel high-level FDIR method called *Distributed Failure Detection* is proposed in this PhD thesis, which is targeted at satellite payload data processing applications. Compared to earlier approaches, it is original because the data processing application is partitioned into several hardware blocks called *stream processors*. These processors communicate with each other via a modern switched fabric network architecture that offers excellent scalability. The Distributed Failure Detection method allows the application of hot redundancy to stream processors. This is made possible by integrating the failure detection mechanisms into the network architecture. As a consequence, redundant stream processors can be distributed throughout the network without restrictions. This can be a real advantage for large systems, in which processing nodes are constantly added, deleted, removed or reconfigured. Although it is later demonstrated that this approach works fine for reconfigurable multi-FPGA systems, it is not limited to those. Since the approach is rather technology-independent, it could even be applied to similar distributed computing systems.

To fill the second gap, a novel *Availability Analysis* method is proposed, which aims at predicting the achievable reliability of a stream processor in a particular radiation environment. In addition, it allows a fair comparison of different FDIR configurations. The proposed approach advances the current state of the art in several aspects. The systematic method is implemented as a set of tools integrated together and could therefore be easily automated. One of the tools is a novel fault injection system, which can emulate SEUs in the configuration memory of a reconfigurable FPGA. Compared to earlier systems, it cannot only find the so-called sensitive bits but is also able to classify them, depending on how the system can recover from failures, which are triggered by upsets in these sensitive bits. By doing so, the achievable reliability of more complex FDIR approaches can also be analysed. Another tool is a novel Block RAM profiling tool, which estimates the number of sensitive RAM bits by analysing the average usage of embedded RAM blocks in simulation. Such a method has not previously been proposed in literature. However, it can significantly increase the prediction precision because the influence of embedded RAM blocks cannot be neglected for many data processing applications.

The proposed FDIR approach as well as all tools required for the Availability Analysis method are implemented in a proof of concept system. Aside from demonstrating the concepts, the system is used for an in-depth analysis of several popular FDIR approaches in terms of power, area and performance overhead. This study is another original contribution to the current state of the art. Again, although similar research results can be found in literature, they are mainly used to emphasise the capabilities of a particular mitigation technique. However, a fair comparison between different techniques is often difficult to draw due to a variety of measurement setups and terminologies used by different research groups.

To complete the picture of the research work, the proposed approaches are all validated in a real radiation environment by irradiating parts of the proof of concept system with accelerated protons. Firstly, it is demonstrated that the FDIR hardware and software components are mature enough to detect and recover from failures in a real radiation environment that causes much higher SEU rates than any solar particle event observed in history. Secondly, it is proven that the Availability Analysis method provides accurate estimates.

The research approaches and results presented in this thesis can be valuable for both scientists and engineers. Firstly, it can serve as a tutorial for the space engineering community since in-depth background information is given for particular FPGA devices that are suitable for data processing applications on board spacecraft. Most notably, the hands-on character of the proposed Availability Analysis method and the practical results gained from the power, area and performance overhead measurements as well as from the irradiation test campaign can contribute valuable information to other space projects. Secondly, the proposed FDIR approach is not only of interest for space engineers but also for researchers concerned with distributed computing in general. Since the approach is rather technology-independent, it can also be abstracted from the FPGA user case and applied to other computing systems.

1.3 OVERVIEW OF THESIS CONTENTS

The thesis is structured as follows. In Chapter 2, the state of the art in this rich research field is surveyed. After a thorough review of recent literature, research gaps are identified and existing mitigation techniques are summarised by providing a decision strategy for researchers and engineers who are novices in this field. In Chapter 3, the theory behind the proposed Distributed Failure Detection method is explained, including a detailed description of all employed FDIR components. The proposed Availability Analysis method is introduced in Chapter 4, including a functional description of the novel fault injection and Block RAM profiling tool. Next, the proof of concept system is described in Chapter 5 by providing detailed information about all hardware and software components. With this proof of concept system, the power, area and performance overhead of several popular FDIR approaches is measured in Chapter 6. Thereafter, Chapter 7 demonstrates how the Availability Analysis method can be applied in practice by predicting the reliability of the proof of concept system in different FDIR configurations and for different example satellite missions. Both the Distributed Failure Detection and the Availability Analysis method are then validated by means of an accelerated proton irradiation test campaign in Chapter 8. Finally, Chapter 9 concludes the PhD thesis and gives an overview of future work.

1.4 PUBLICATIONS

Some of the results of this thesis have been reported in the following publications:

JOURNAL PAPERS

- F. Siegle, T. Vladimirova, O. Emar, and J. Ilstad, "Mitigation of radiation effects in SRAM-based FPGAs for space applications," in *ACM Computing Surveys*, 2014
- F. Siegle, T. Vladimirova, O. Emar, and J. Ilstad, "Availability analysis for satellite data processing systems based on SRAM FPGAs," in *IEEE Transactions on Aerospace and Electronic Systems*, 2015 (under review)

CONFERENCE PAPERS

- F. Siegle, T. Vladimirova, O. Emar, and J. Ilstad, "Adaptive FDIR strategy for FPGAs hosting partial reconfigurable modules," in *Workshop on Reconfigurable Computing (WRC), HiPEAC Conference*, Jan. 2013
- F. Siegle, T. Vladimirova, O. Emar, and J. Ilstad, "Adaptive FDIR framework for payload data processing systems using reconfigurable FPGAs," in *Proc. of 8th NASA/ESA Conference on Adaptive Hardware and Systems*, June 2013
- F. Siegle, T. Vladimirova, O. Emar, and J. Ilstad, "New voter design enabling hot redundancy for asynchronous network nodes," in *Proc. of 9th NASA/ESA Conference on Adaptive Hardware and Systems*, July 2014
- F. Siegle, T. Vladimirova, O. Emar, and J. Ilstad, "Fault detection, isolation and recovery techniques for SRAM-based multi-FPGA systems," in *Proc. of the Military and Aerospace Programmable Logic Devices (MAPLD) Workshop*, May 2014
- F. Siegle, T. Vladimirova, O. Emar, and J. Ilstad, "FDIR techniques for payload streaming applications using SpaceWire-based networks," in *Proc. of the International SpaceWire Conference*, 2014
- F. Siegle, T. Vladimirova, C. Poivey, and O. Emar, "Validation of FDIR strategy for spaceborne SRAM-based FPGAs using proton

radiation testing," in *Proc. of the Conference on Radiation Effects on Components and Systems (RADECS 2015)*, 2015

2

CHAPTER 2: LITERATURE SURVEY

2.1 INTRODUCTION

Electronics on board modern spacecraft comprise a considerable number of FPGA devices. Although these devices were mainly used to implement rudimentary glue logic in the early days, they enable far more complex operations today. Regardless of the application such as science, Earth observation or military surveillance, a trend to ever increasing payload data volumes can be observed. Thus, data processing in space can be essential for some missions as payload data downlinks can be too slow to transmit these growing data volumes, even if data compression techniques are applied.

Many payload data processing applications benefit from an efficient implementation in hardware using programmable logic devices. Modern SRAM-based FPGAs offer huge amounts of logic resources, allow fast clocking and can quickly be reconfigured which makes them ideal platforms for the implementation of such algorithms. They are, however, prone to radiation effects in space because the state of their memory cells can be flipped due to single and multiple event upsets caused by radiation. Hence, design techniques to mitigate radiation effects must be applied to these devices. In the past ten years, research on such mitigation methodologies established its own rich field, which is thoroughly surveyed in this chapter.

The chapter is structured as follows. Section 2.2 discusses radiation effects in space as well as their effects on SRAM-based FPGAs. Then, in Section 2.3, terminology, failure modes and mitigation techniques are outlined. In Section 2.4, mitigation techniques applied during runtime operation are reviewed dividing this huge field into three areas: techniques aimed at the (i) configuration memory, (ii) user logic and (iii) embedded RAM blocks. A brief survey of methodologies that can be applied during design time is then given in Section 2.5. Then, Section 2.6 covers the simulation, emulation and analysis of radiation effects, including accelerated radiation testing and fault injection. Section 2.7 is dedicated to purpose-built hardware platforms, which have been used in research projects on SRAM-based FPGAs for space

applications. Section 2.8 provides a summary of the reviewed techniques as well as design recommendations. In Section 2.9, the state of the art is analysed to identify research gaps that can be filled with innovations proposed in the course of this PhD work. Finally, Section 2.10 concludes the chapter.

2.2 RADIATION EFFECTS IN SRAM-BASED FPGAS FOR SPACE

2.2.1 *Sources of Radiation Effects*

The space radiation environment comprises a large range of energetic particles with energies from several keV up to GeV and beyond. The main elements are [15, 16]:

- Trapped radiation: Energetic electrons and ions are magnetically trapped in the so-called Van Allen radiation belts which extend from 100 km to 65,000 km and consist mainly of electrons up to a few MeV and protons of up to several hundred MeV energy. The Earth's magnetic field is not symmetrical, leading to local distortions. One important distortion is known as the South Atlantic anomaly. Spacecraft passing this area are exposed to an increased level of radiation.
- Galactic cosmic rays: High-energy charged particles which enter the solar system from outside and which are composed of protons, electrons and fully ionized nuclei.
- Solar energetic particles during solar flares and coronal mass ejections: High-energy particles which are encountered in interplanetary space and close to Earth and which are seen in short bursts associated with other solar activity. The duration of such bursts can be a few hours up to several days. They consist of protons, electrons and heavy ions in the energy range of a few tens of keV to GeV and beyond. The particles of the regular solar wind have rather low energies and are thus less of a concern for electronic devices in space.

In addition, secondary radiation is generated by the interaction of energetic particles with materials. One example is *bremsstrahlung*, a high-energy electromagnetic radiation that is caused by the deceleration of a charged particle in materials.

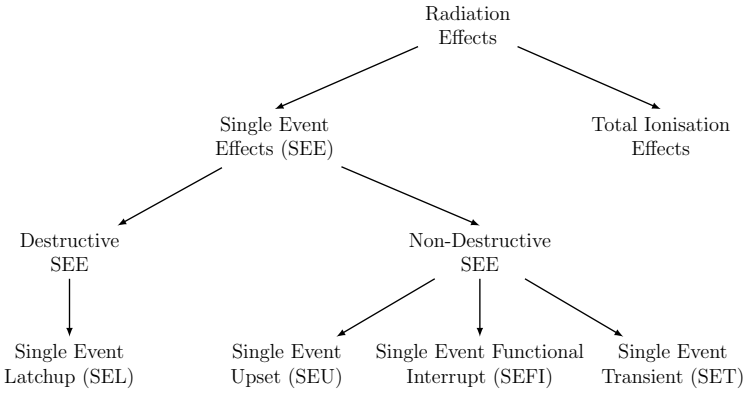

Figure 1: Common radiation effects that must be mitigated in SRAM-based FPGAs

2.2.2 *Radiation Effects*

An overview of common radiation effects that must be mitigated in SRAM-based FPGAs is given in Figure 1. The main effects are:

- TID Effect: Ionisation of electronic components is caused by electrons, protons and bremsstrahlung and leads to a degradation due to increasing leakage currents and other effects [17]. Processes that cause ionisation are based on photon interaction and include the photoelectric effect, compton effect and pair production, all leading to the production of free electrons and hole-electron pairs [18]. The accumulation of these effects is called TID and is usually measured in krad with $1\text{rad} = 10^{-2}\text{Gy} = 6.24 \cdot 10^{7} \frac{\text{MeV}}{\text{g}}$ [19]. For space-qualified Virtex-4QV and Virtex-5QV devices, the TID is of no concern since the dose is guaranteed to be 300 krad for Virtex-4QV devices [20], respectively 1 Mrad for Virtex-5QV devices [21].

- SEL: A potentially destructive SEE that can trigger parasitic PNPN thyristor structures in a device [17]. Similar to the TID effect, SELs are of no concern for Virtex-4QV and Virtex-5QV devices since both devices have a guaranteed latchup immunity to $\text{LET} > 100\text{MeV} \cdot \text{cm}^2 \cdot \text{mg}^{-1}$ [20, 21].

- SEU: This class of SEE is a *soft error* that changes the state of a bistable element. It is triggered by heavy ions and protons and results from ionisation by a single energetic particle or the nuclear reaction products of an energetic proton. The ionisation induces a current pulse in a p-n junction whose charge may exceed the critical charge which is required to change the logic

state of the element. As a result, the value of a memory bit can be flipped [15]. SEU is the most common effect for SRAM-based FPGAs as it may affect the configuration memory as well as memory cells that are used as part of the user logic (flip-flops, embedded RAM).

- Single Event Functional Interrupt (SEFI): This class of SEE interferes with the normal operation of the FPGA and is thus typically used to classify failures that affect the circuits needed to operate the FPGA. So far, six types of SEFIs have been identified for Virtex-4QV devices [3]:
  - Power-On-Reset (POR) SEFI: results in a global reset of all internal storage cells and the loss of all program and state data.
  - SelectMAP (SMAP) SEFI: results in loss of either read or write capability through the SelectMAP interface.
  - Frame Address Register (FAR) SEFI: results in the frame address register continuously incrementing.
  - Global Signal SEFI: results in disruption of global signals like Global Write Enable, Global Drive High etc.
  - Readback SEFI: occurs when a portion of readback data has been upset and can lead to a false-positive detection of a SMAP SEFI.
  - Scrub SEFI: causes corruption of the data stream being downloaded to the device.
- Single Event Transient (SET): This class of SEE is a momentary voltage or current disturbance which may propagate through subsequent circuitry and eventually manifests as SEU once it reaches a latch or other memory elements [22].

2.2.3 *Single Event Effects Rates*

SEU rates in FPGAs depend on the particular device component in which they occur (see Section 2.3.2 below). The mitigation strategy for Virtex-4QV FPGAs must mainly take into account single event effects, as these devices are tolerant to accumulated ionisation and SELs. In contrast, Virtex-5QV devices are radiation hardened by design. This was achieved by replacing the configuration memory and flip-flop cells by dual-node counterparts that require charge collection in at

Table 2: Example SEFI and SEU rates for a Virtex-4QV SX55 [3]

Component	Unit	LEO	GEO
All SEFIs	$^{\text{Device·Years}}/_{\text{Events}}$	36	103
User flip-flops	$^{\text{Upsets}}/_{\text{Device·Day}}$	0.0702	0.0387
Block RAM Memory	$^{\text{Upsets}}/_{\text{Device·Day}}$	4.05	4.49
Configuration Memory	$^{\text{Upsets}}/_{\text{Device·Day}}$	7.56	4.28

least two active nodes before an upset can occur. Furthermore, all flip-flop inputs are now protected by SET filters and Triple Modular Redundancy (TMR) is applied to control circuitry and registers [23].

Static and dynamic cross-sections for most FPGA blocks with regards to the Virtex-4QV family, can be found in [3, 24]. Using these cross-sections, SEU and SEFI rates can be calculated for a particular design and orbit. For European space projects the necessary calculation methods are standardised in [16] and [17].

In [3], SEU and SEFI rates for several orbits in quiet solar maximum conditions were calculated using the CREME96 model. For illustration, rates for two orbits are given in Table 2. The first one is a Low Earth Orbit (LEO) at 800 km altitude with an inclination of 22.0°, the second one is a Geostationary Earth Orbit (GEO) at 36,000 km. The FPGA type is a XQR4VSX55 and it is assumed that all memory cells are used, i.e. the upset rates per bit-day are scaled to the whole device. It can be seen that the likelihood of SEFIs is low, with approximately one SEFI every 36 years in LEO and every 103 years in GEO.

Assuming that all flip-flop cells are used, the chance of an upset in these elements is far below 0.1 upsets per device-day. In contrast, if a design heavily utilises Block RAM (BRAM) blocks (in this example all blocks are used), the probability of an upset is more than 400 times higher than for a flip-flop upset due to the high ratio of BRAM cells to flip-flop cells. For the configuration memory cells the ratio is even larger: In LEO more than 7.5 upsets can occur per device-day. It is, however, assumed that all configuration memory cells are utilised which is unrealistic for a real design.

The results above show that mitigation techniques must mainly focus on configuration memory and Block RAM upsets. Although SEFIs occur only rarely, they can necessitate an undesired full reconfiguration and must therefore be mitigated as well as possible. In contrast,

a mitigation strategy for flip-flops may not be necessary for some applications.

In 2012, Quinn et al. presented first on-orbit results for Virtex-4QV FPGAs [25], collected from an experimental payload launched by Los Alamos National Laboratory. The system comprises two Virtex-4 FPGAs running the same digital signal processing application. The mitigation strategy is based on TMR in combination with scrubbing. Using fault injection experiments and the CREME96 model, an observable output error rate of approximately one in 15 to 25 days was predicted before launch. This rate is based on a calculated configuration memory upset rate of 68 to 89 $^{\text{SEU}}/_{\text{device·day}}$.

The on-orbit results are surprising: Firstly, the measured upset rate per unit is much lower than predicted (19 $^{\text{SEU}}/_{\text{unit·day}}$). Secondly, the only two measurable output errors were triggered by SEUs in bit locations which could not be predicted by fault injection before. Thirdly, a SelectMAP SEFI was observed, although such a failure should only occur very rarely according to worst case predictions. Finally, the authors were able to observe atypical events where many bits in one single frame were corrupted all at the same time.

The authors assume that the measured upset rate is artificially low due to the shielding of the spacecraft and the duty cycle of the device. It was further found that 8.42% of SEU events are actually Multiple Bit Upsets (MBUs), although the vast majority has a size of only two bits. 78% of the SEUs occurred in Configurable Logic Blocks (CLBs), followed by ca. 15% in BRAM interconnect and ca. 6% in Input Output Blocks (IOBs).

2.3 OVERVIEW OF RADIATION MITIGATION TECHNIQUES

2.3.1 *Terminology*

Several techniques can be applied during the design process to mitigate soft errors in digital circuits. A classification of these techniques is presented in Section 2.3.3 below, which makes use of the terminology introduced in the NASA Fault Management Handbook [26]. Although targeting flight systems in general, this terminology proves to be well suited to describing soft error mitigation techniques for FPGAs too.

A common terminology to describe an abnormal state of a system includes the three terms: fault, error and failure. Although several

standards define these terms slightly differently, a fault is usually understood as the cause of an error and an error as the cause of a failure. For instance, in functional safety standard ISO 26262, a fault is defined as an "abnormal condition that can cause an element or an item to fail". The error is defined as 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 defined as the "termination of the ability of an element, to perform a function as required".

According to the Fault Management Handbook, failures can either be prevented or tolerated. In the first case, actions are taken to avoid failures either at design time or run time. The Design-Time Fault Avoidance includes "design function and FM [fault management] capabilities to minimize the risk of a fault and resulting failure" whereas Operational Failure Avoidance "predicts that a failure will occur in the future and takes action to prevent it from happening". With failure tolerance, failures are either accepted or mitigated. Failure Masking techniques "allow a lower level failure to occur, but mask its effects so that it does not affect the higher level system function". Failure Recovery techniques "allow a failure to temporarily compromise the system function, but respond and recover before the failure compromises a mission goal". Finally, Goal Change strategies "allow a failure to compromise the system function, and respond by changing the system's goals to new, usually degraded goals that can be achieved".

In the following, erroneous FPGA output is seen as a failure. Although the failure is always caused by a fault, a fault does not necessarily lead to a failure. In an FPGA circuit design, such a fault could be for example a flipped bit in a flip-flop or a re-programmed logical operation due to a falsified look-up table. In any case, only if the faulty resource is actually used in the design the associated fault will finally lead to a failure.

2.3.2 *Failure Modes in SRAM-based FPGAs*

An FPGA model, commonly found in literature [27], which is suitable for illustration of different fault and failure modes of SRAM-based FPGAs is shown in Figure 2. SRAM-based FPGAs comprise a configuration memory layer that stores the configuration of the FPGA in SRAM memory cells and a user logic layer on which the actual cir-

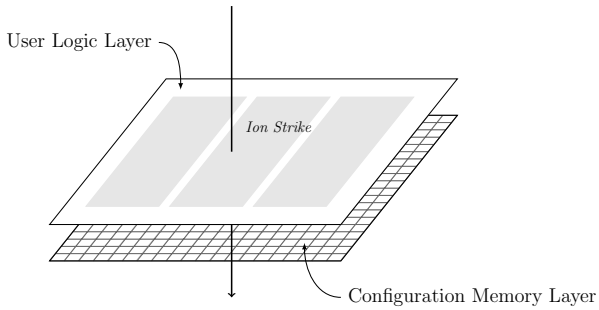

Figure 2: Model of an SRAM-based FPGA

cuit is implemented. A typical circuit utilises sequential and combinational logic elements and often accesses embedded BRAM and/or DSP blocks. User flip-flops and other user memory resources as well as the DSP blocks are physically present, whereas combinational logic gates are realised with Look-Up Tables (LUTs) within CLBs.

The configuration bits on the configuration memory layer control the resources on the user logic layer, including the wiring between the resources, the content of the LUTs and the configuration of the CLB, BRAM, DSP and IOB blocks.

If an ion hits the FPGA, it can affect memory resources (i) on the configuration memory or (ii) on the user logic layer. In both cases, upsets can be seen as faults which *may* lead to a failure. And in both cases, the system can fortunately recover from such failures because affected memory cells can be updated with correct values. Since the configuration bits control "really everything" [27], the configuration memory is the main concern of most mitigation strategies. However, although more than 60 percent of the configuration bits are used to control routing resources, only 10 to 20 percent of routing resources are used in a typical design [28]. The ratio between used configuration bits and user flip-flops bits is, however, usually still so high that flip-flop upsets account for only a few percent of all upsets. And obviously, configuration bit upsets can lead to much more unpredictable behavior than flip-flop upsets. In contrast to user flip-flops, Block RAM upsets can be as much of a concern as configuration memory upsets if large amounts of these resources are utilised in a design.

A fault in the configuration memory may lead to a failure in case the affected configuration bit controls a resource which is utilised by the design. In Xilinx terminology, configuration bits can be classified as Essential and Critical Bits [29]. Essential Bits are the subset of configuration bits which are responsible for resources of the design. Thus, a fault affecting an Essential Bit *may* lead to a failure. Because

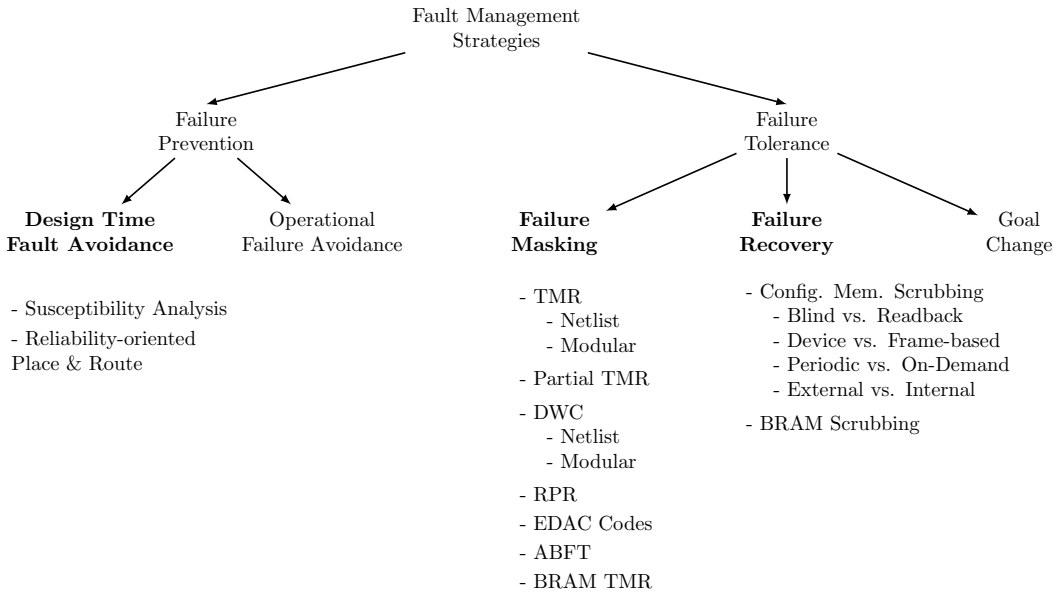

Figure 3: Classification of fault management strategies and corresponding mitigation techniques

not every resource of a design is used by the application non-stop, only faults in a subset of the Essential Bits, also referred to as Critical Bits, are guaranteed to manifest as failure.

A fault in a user flip-flop can lead to a failure if its value is used by subsequent circuitry. Although the failure can propagate through the system until it becomes measurable at the output, it is often only of transient nature. If the flip-flop is used in state-dependent logic, however, a failure can be 'trapped' in a feedback loop until the logic is reset to a known (initial) state. For instance, if a bit of a counter register is flipped, the counter 'jumps' and the output is permanently falsified.

A fault in a Block RAM cell can lead to a failure with the next read access. Often, the memory is not immediately accessed and the manifestation of the failure is delayed.

2.3.3 *Classification of Mitigation Techniques for SRAM-based FPGAs*

Figure 3 shows an overview of fault management strategies, classified according to the aforementioned terminology, together with the corresponding mitigation techniques surveyed in this chapter.

During run time, failure masking techniques can be used to tolerate failures. Failure masking is usually achieved by redundancy. Most

commonly, spatial redundancy is applied, for instance TMR, Partial TMR, Duplication with Compare (DWC) or Reduced Precision Redundancy (RPR). Alternatively, information redundancy techniques can be used to detect and mask failures in certain types of circuits, for example Error Detection and Correction (EDAC) codes or Algorithm Based Fault Tolerance (ABFT).

Aside from failure masking, failure recovery techniques can be used during run time too. Failure recovery is usually done by refreshing the memory, which is often referred to as scrubbing.

During design time, several techniques can help to avoid faults in advance, including tools for the susceptibility analysis (e.g. for the quantification of sensitive configuration bits) and tools which place and route a circuit design in a reliability-oriented way.

2.4 MITIGATION DESIGN TECHNIQUES AIMED AT RUN-TIME FAILURE TOLERANCE

In this section a review of the rich field of failure tolerance techniques that can be applied during run-time, targeting both the configuration memory and the user logic is presented. Different scrubbing approaches are outlined with regards to the configuration memory. Besides implementation specific differences (Blind vs. Readback Scrubbing, Device vs. Frame-Oriented Scrubbing, External vs. Internal Scrubbing), two fundamentally different concepts found in literature, are discussed too. The first concept combines periodic scrubbing with a low-level redundancy approach whereas the second concept implements an FDIR approach in which the configuration memory is only repaired once a failure has been detected in user logic. For the user logic, different redundancy concepts are surveyed. Again, it turns out that the concepts presented in literature can roughly be divided into two categories. The first type of spatial redundancy is applied to the netlist of the circuit and is thus a quite low-level approach. The second type is a modular redundancy approach in which whole hardware blocks are operated in hot redundancy.

2.4.1 *Configuration Memory*

Single and multiple bit upsets in the configuration memory of SRAM-based FPGAs can be mitigated by periodically writing a known to be correct bitstream to the device. This technique is often referred to as

scrubbing and several types of implementations can be found in research literature and application notes. In the following, the different methodologies and architectures are classified using a similar terminology as introduced in [30, 31] including:

- Blind vs. Readback Scrubbing.
- Device vs. Frame-Oriented Scrubbing.
- Periodic vs. On-Demand Scrubbing.
- External vs. Internal Scrubbing.

2.4.1.1 *Blind vs. Readback Scrubbing*

The most basic methodology is blind scrubbing where the configuration memory is periodically updated with a known to be good copy of the original bitstream. This copy which is sometimes referred to as the 'golden copy', is stored in an external, radiation hardened memory. An external or internal configuration controller controls the download of the bitstream via one of the configuration interfaces of the FPGA. Using the classification shown in Figure 3, blind scrubbing can be described as an operational failure avoidance methodology because faults are handled in a preventive manner without any knowledge about the current health state of the system.

One concern that is sometimes raised in connection with blind scrubbing is the fact that the configuration controller gains write access to the configuration memory even if there is no need for scrubbing. Since the configuration interface is prone to SEFIs, a bitstream download can be affected by radiation effects, potentially leading to a corrupted design. Therefore, Xilinx recommends a SEFI detection before each write access that includes a FAR check and a status and control register check [28].

To further minimise the risk of a corrupted bitstream download, the readback feature of SRAM-based FPGAs can be utilised for scrubbing. Using one of the configuration interfaces, bitstreams cannot only be written to the device but also read back during operation. With this capability, unnecessary write accesses to the configuration memory can be avoided during scrubbing: Before writing a correct bitstream to the device, the current bitstream is read and checked for upsets. Only if upsets are detected, the correct bitstream is eventually written. Such a scrubbing methodology can be identified as a failure recovery technique. Two possible detection mechanisms are commonly

used: The first one is based on comparison and relies on golden bitstream copies. The current bitstream is read back from the FPGA and compared to the golden copy, either by bit-wise comparison or more simply, by calculating a Cyclic Redundancy Check (CRC) checksum during readback which can then be compared with the CRC value of the golden copy (later referred to as CRC readback scrubbing). If a mismatch is detected, the golden copy is used to overwrite the current bitstream.

The second detection mechanism is based on information redundancy and uses the Error-Correcting Code (ECC) bits which are embedded into each configuration frame. This Single Error Correction and Double Error Detection (SECDED) code allows the detection of single and double bit upsets and the correction of single bit upsets. For multiple bit upsets with more than two wrong bits, the syndrome value is indeterminate [32]. During readback, a syndrome value is calculated by an ECC logic that must be initiated as a user primitive called `FRAME_ECC_VIRTEX4` for Virtex-4 devices and `FRAME_ECC_VIRTEX5` for Virtex-5 devices respectively [33]. The syndrome value does not only identify upsets but can also localise single upsets. Hence, two possible failure recovery methodologies can be combined with the ECC logic: Either the erroneous bit is flipped and the corrected bitstream is written back to the device (later referred to as ECC readback scrubbing) or the whole bitstream is overwritten with a golden copy from memory.

A methodology that allows the detection and correction of multiple bit upsets using a custom-built EDAC core is presented in [34]. The authors divide a configuration frame into several data segments and interleave the bits of these data segments. Then, an EDAC check code is calculated for each segment. Since adjacent memory cells are distributed over several data segments, multiple bit upsets can be detected and corrected. A recent work that advances this concept is proposed in [35]. Here, the process of detecting multiple bit upsets and correcting them is separated. The detection is done using a novel lightweight error detection coding technique called Interleaved Two Dimensional Parity whereas the correction utilises so-called erasure codes as can be found in reliable storage devices and similar applications.

Starting with the Virtex-5 architecture, an internal readback CRC logic allows a continuous and automatic readback in the background [33]. In the first readback round, a golden CRC checksum is calculated

Table 3: Summary: Blind scrubbing vs. readback scrubbing

Methodology	Attributes
Blind Scrubbing	• Simple implementation. • Robust for MBU mitigation. • Unnecessary write accesses.
Readback & Comparison	• Comparator necessary. • Robust for MBU mitigation. • Reduced risk of corrupted download.
Readback & EDAC	• Rather complex implementation. • Cannot cope with more than two upsets per frame. • Reduced risk of corrupted download.

which is later used to compare the CRC values of the subsequent rounds to. Once a mismatch has been detected, dedicated user logic can initiate a reconfiguration of the device or a bitstream repair using the ECC logic [36]. A summary of Blind and Readback Scrubbing approaches is given in Table 3.

2.4.1.2 *Device vs. Frame-based Scrubbing*

All scrubbing methodologies mentioned in the last paragraph can use different bitstream sizes. However, the configuration memory is typically scrubbed with a full bitstream or on a frame by frame basis. The first case, sometimes also referred to as device-based scrubbing, requires a rather simple implementation. Except for the modified header information, the bitstream can directly be downloaded from a memory to the configuration interface. One drawback of this

Table 4: Summary: Device-based vs. frame-based scrubbing

Methodology	Attributes
Device-based	• Simple implementation. • Scrubbing SEFIs may affect the whole design.
Frame-based	• Increased implementation complexity. • Decreased scrubbing speed. • Scrubbing SEFIs can only affect one frame.

solution is the susceptibility of the configuration interface to SEFIs. If such an upset occurs during download, the whole design is likely to become corrupted. Frame-based scrubbing requires a more complex configuration controller implementation because each frame must be prepared before download. But, the benefit of this approach is the possibility to isolate the effects of a SEFI to a single frame. Aside from the increased complexity of implementation, the scrubbing speed is decreased too. First, a SEFI check must be done before downloading each frame. Secondly, each frame bitstream comes with an overhead due to its header. Finally, after each frame a dummy frame must be written to flush the pipeline [28].

In some applications increased scrubbing speed is desired. This is especially true for applications in which scrubbing is used as the only mitigation technique. In [37], such a 'low-cost' strategy, based on an idea presented in [38], is proposed. The authors point out that many configuration frames are scrubbed although they contain no or only a small number of essential bits. As a consequence, they propose to constrain the placement of the design in such a way that the number of frames with essential bits is minimised. Then, the frame-based scrubber must only take this subset of frames into account. A summary of Device and Frame-based Scrubbing approaches is given in Table 4.

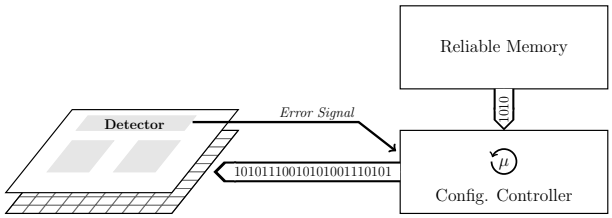

Figure 4: On-demand scrubbing

2.4.1.3 *Periodic vs. On-Demand Scrubbing*

In many designs, the scrubbing process is independent of other mitigation techniques. Then, the configuration memory is periodically scrubbed for upsets with a fixed scrubbing rate.

Alternatively, the scrubbing process can also be triggered by a failure detection mechanism. Such a methodology can be advantageous in systems where continuous scrubbing is unwanted. Eventually, the availability of a system depends on the time a faulty component remains unrepaired. This time can be minimised by either increasing the scrubbing frequency or by implementing a mechanism that can trigger a repair process immediately after failure detection. With the aid of stochastic models, Siegle et al. showed in [9] that in principle, on-demand scrubbing always maximises the availability.

Depending on the implementation, on-demand scrubbing can be power saving because the scrubbing logic is only active if required. Becker et al. investigated in [39] the power consumption of Virtex-II devices during reconfiguration. Although the authors use a LZSS decompression core during reconfiguration, the power consumption is increased by only 95 mW. Nevertheless, the avoidance of additional resource overhead is always beneficial, especially for systems where failure detection mechanisms are implemented anyway.

In the literature, on-demand scrubbing is mainly mentioned in connection with systems utilising dynamic partial reconfiguration. In many proposed systems, spatial redundancy, e.g. TMR, is implemented by using redundant reconfigurable modules and a majority voter within the static area. Due to the physical separation of the reconfigurable modules, on-demand scrubbing can be advantageous here. The majority voter can easily be designed as a failure detection mechanism which is able to identify a faulty module and which can trigger a scrubbing process on-demand, targeting only the faulty component.

Paulsson et al. presented such a system in [40] for Virtex-II devices. The authors use so-called Dynamic TMR or Hardware Test Benches

as failure detection mechanisms and reconfigure a partition only if a failure has been detected. Researchers at University of Arizona proposed similar mechanisms for their SCARS system that is based on Virtex-5 devices [41]. Here, a faulty reconfigurable module is only scrubbed after a failure has been detected by software routines. Jacobs et al. propose a similar approach in [42]. Again, failures in reconfigurable modules are detected by voters or comparators and scrubbing is triggered only for the faulty module on-demand. Straka and his colleagues at University of Brno [43] also work on a fault tolerant framework for SRAM-based FPGAs. Very similar to the already mentioned approaches, a so-called Generic Partial Reconfiguration Controller receives error signals from reconfigurable modules and triggers on-demand scrubbing if required. Azambuja et al. also use majority voters as failure detection mechanisms and scrub a faulty reconfigurable module only after a failure has been detected [44, 45]. The authors emphasise the increased repair speed compared to a full reconfiguration. In [46], Iturbe et al. propose a fault management strategy in which they combine on-demand blind scrubbing, triggered by a majority voter as failure detection mechanism, with ECC readback scrubbing.

A methodology that aims at speeding up the on-demand scrubbing process is presented in [47]. The authors analyse the statistical distribution of sensitive bits within a partial bitstream. Instead of starting the scrubbing process from the first byte position of the bitstream, it is started from the one frame for which the authors calculated that this start position minimises the Mean Time to Recover (MTTR). For a set of benchmark circuits, an average MTTR reduction of 30% was achieved. A methodology with a similar aim is presented in [48]. The authors partition a circuit design and apply a specific redundancy scheme (like DWC or TMR) to each partition. If one of these partitions is detected to be faulty, it is scrubbed by an external reconfiguration controller on demand. The authors developed an algorithm which optimises the floorplanning of the different partitions to find an optimal solution in terms of reconfiguration time, area and performance overhead. Results for a set of example circuits suggest that the reconfiguration time can heavily be reduced although this reduction is at the cost of an increased area and performance overhead. A summary of Periodic and On-Demand Scrubbing approaches is given in Table 5.

Table 5: Summary: Periodic vs. on-demand scrubbing

Methodology	Attributes
Periodic	• Mean Time To Recover (MTTR) depends on scrubbing rate. • No failure detection mechanisms on user logic layer necessary. • Possibly increased power consumption.
On-Demand	• MTTR is minimised. • Partial scrubbing is possible. • Access to configuration memory is minimised. • Failure detection mechanisms on user logic layer necessary.

2.4.1.4 *External vs. Internal Scrubbing*

The scrubbing logic can be implemented internally or externally. From the available configuration interfaces, the SelectMAP interface is commonly used for external scrubbing due to its high throughput rates. The Internal Configuration Access Port (ICAP), internal counterpart to SelectMAP, can be used if the scrubbing logic is implemented on user logic layer. Internal scrubbing is sometimes seen as a 'low-budget' solution because it does not necessitate an external configuration controller and a memory for the golden bitstream copies. It can be argued, however, that in most space applications a radiation hardened supervisor as well as reliable memory for the initial bitstream configuration is available anyway.

External scrubbing via the SelectMAP interface is commonly seen as the more robust approach and is also recommended by Xilinx [28]. Melanie Berg and other researchers at NASA come to similar conclusions in [49] where the authors compare an external blind scrubber to an internal ECC readback scrubber by Xilinx. The internal scrubber is based on a PicoBlaze microcontroller and its design was published in the no longer available application note [50]. Using heavy-ion SEE radiation testing, it was found that the external scrubber was always recoverable without the need for a reset or power cycle whereas the internal scrubber was never recoverable. Thus, the internal scrubber consistently reached a state where it could not operate anymore, either because of MBUs which cannot be handled by the scrubber or because the scrubber itself was hit by ions.

Heiner et al. from Brigham Young University improved the fault tolerance of the same Xilinx scrubber design by applying TMR and Block RAM scrubbing [51]. In radiation tests it was found that the improved scrubber performs much better but still, in more than 45% of all tests the design failed at some point, requiring a subsequent full reconfiguration of the device. The authors assume that the missing ability of ECC readback scrubbers to repair MBUs was the main reason for this behaviour.

Ebrahim et al. from the University of Edinburgh also work on a fault-tolerant ICAP controller in the course of their R3TOS system [52]. The controller is based on Xilinx' `XPS_HWICAP` core. The controller is not only used for scrubbing but also for partial reconfiguration. Similar to the ICAP controller described above, the scrubber is an ECC readback scrubber. To improve its fault-tolerance the authors apply spatial redundancy but instead of applying TMR to the whole

Table 6: Summary: External vs. internal scrubbing

Methodology	Attributes
External	• Robust. • Radiation hardened external controller and memory needed.
Internal	• In case of ECC readback scrubbing, MBUs cannot be repaired. • No external controller and no golden bitstream copies necessary.

controller, only a so-called Recovery Module is triplicated. This module, on the other hand, is able to gain access to the ICAP interface only for the sake of recovering the controller from failures. A summary of External and Internal Scrubbing approaches is given in Table 6.

2.4.1.5 *Integration with Dynamic Partial Reconfiguration*

Most scrubbing approaches described in literature assume a static user design. If dynamic partial reconfiguration is used as part of the normal operation, however, e.g. to time-share chip area by swapping different modules during runtime, the reconfiguration and scrubbing process must be somehow orchestrated because only one of them can gain access to the configuration interface at the same time. Furthermore, if blind scrubbing or CRC readback scrubbing is used, the golden bitstream must be kept updated after each partial reconfiguration to mirror the currently running design.

One approach to overcome these problems is described by Heiner et al. in [30]. The authors use a CRC readback scrubber as described earlier. Instead of downloading the bitstream of a reconfigurable module and updating the golden bitstream afterwards, the authors suggest to simply integrate the bitstream of the reconfigurable module into the golden bitstream. During the next scrubbing cycle the scrubber detects a discrepancy between the golden bitstream and the bit-

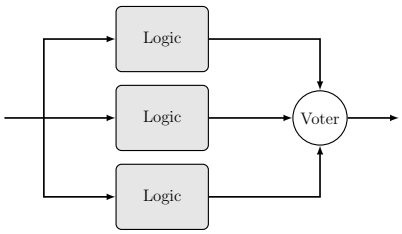

Figure 5: Triple Modular Redundancy

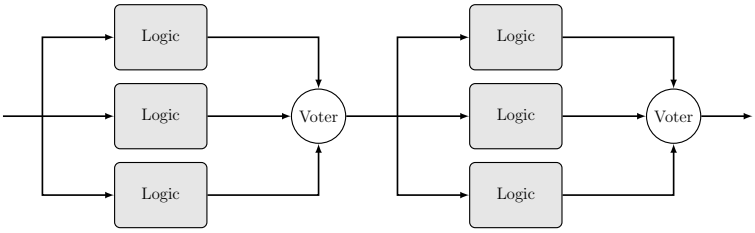

Figure 6: Triple Modular Redundancy with two partitions

stream which has been read back from the device due to mismatching CRC sums. As a consequence, it will then 'repair' the bitstream by writing the updated frames to the device.

2.4.2 *User Logic*

While failure recovery takes mainly place on configuration memory layer, failure masking is implemented on user logic layer using some kind of redundancy. Most commonly, spatial redundancy is used but also information and temporal redundancy can be found for specific components.

2.4.2.1 *Spatial Redundancy*

By far the most common form of spatial redundancy is TMR. In this approach, all components of a circuit are triplicated as depicted in Figure 5 and a majority voter is placed at the end which chooses the correct output.

To decrease the susceptible area the circuit can further be partitioned by adding additional voters as can bee seen in Figure 6. The possible increase of availability is discussed by McMurtrey et al. in [53] using Markov chains. The authors show that the reliability is indeed increased because the area of each circuit stage and therefore the chance that more than two redundant circuit stages fail is decreased.

Since the voter is a single point of failure it is usually triplicated too. As mentioned earlier, upsets affecting feedback loops, e.g. counter

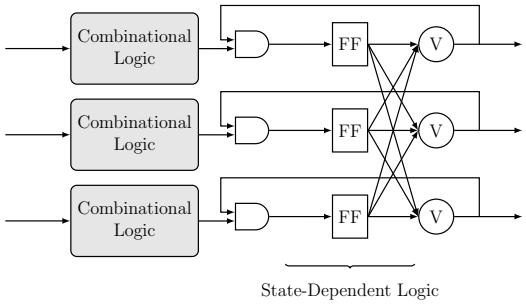

Figure 7: X - Triple Modular Redundancy: Voters are placed in the feedback paths of state-dependent logic to allow automatic re-synchronisation

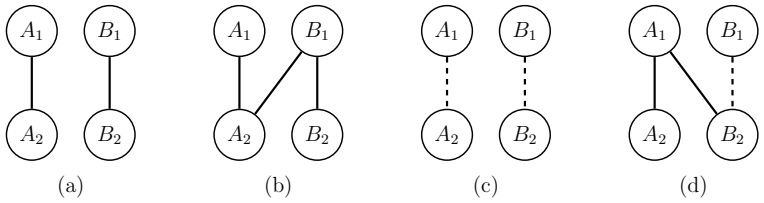

Figure 8: Modes of possible SEU induced effects [1]

or state machines, can be problematic because the failure is trapped in the loop. To overcome this problem voters can be placed inside feedback loops. This technique, sometimes also referred to as XTMR (Xilinx TMR) [54, 55], synchronises the flip-flops automatically after repair, see Figure 7.

Most commonly, TMR is applied to the netlist of a circuit using automatic insertion. Several commercial and academic software tools are available, including the TMRTool by Xilinx [56], Precision Hi-Rel by Mentor Graphics [57] and Synplify Premier by Synopsys [58]. A notable free collection of tools is the BYU EDIF Tool suite, developed at Brigham Young University [59].

Researchers at Politecnico di Torino found analytically that TMR protected circuits are still prone to SEUs because in some cases one single configuration bit upset can lead to multiple failures on user logic layer, invalidating the TMR approach [1]. Logic blocks inside the fabric of the FPGA are interconnected via switch boxes which are built from Programmable Interconnect Points (PIPs). The authors found that one single configuration bit can control two or more PIPs and they identified three possible modifications caused by one SEU, as depicted in Figure 8. Given a pair of connections (a), a short between the connections can occur (b), both connections can be opened (c) or a bridge between the connections can be created (d). If the connections belong to two redundant circuits of a TMR system, the voter

will choose a wrong or falsified output. For Virtex-II devices, this failure mode which is sometimes also referred to as Domain Crossing Error (DCE), was partly confirmed by Quinn et al. [60] by fault injection experiments, although the authors point out that SEU induced "DCEs are possible when TMR is incompletely applied to a design, but they appear to be rare otherwise". However, the authors found that TMR can even be defeated by multiple bit upsets with two or more bits. Using a stochastic model, they predict a worst case probability for DCEs of 0.36% for Virtex-II devices and up to 1.2% for Virtex-5 device.

One obvious drawback of TMR is the large area and thus power overhead that can exceed more than 200%. To decrease the overhead of TMR, several alternatives were proposed in literature. One example is Partial TMR as discussed by Pratt et al. in [61,62]. The basic idea is to apply TMR only to feedback paths and optionally to their inputs to avoid so-called persistent errors [63] in state-dependent logic. By doing so, only failures with a transient nature can occur. The authors demonstrated for a DSP Kernel design that the number of persistent bits decreased by 63% if only the feedback is triplicated at the cost of 26% hardware overhead. By applying Partial TMR to feedback paths and their inputs, the persistent bits were reduced by two orders of magnitude at the cost of 40% hardware overhead. This Partial TMR approach is part of the already mentioned TMR Tool by Brigham Young University.

Another drawback of TMR is its strong impact on the performance of a circuit, especially if the circuit contains many TMR partitions. For instance, Kastensmidt et al. analysed the performance of a digital FIR filter design in [64]. The implementation without TMR could achieve a performance of 154 MHz, whereas the performance of the TMR version with a maximum number of possible partitions dropped down to 123 MHz.

A less common form of spatial redundancy is DWC where a circuit is duplicated and the output of the redundant circuits is compared by a comparator. Naturally, this mechanism is only able to detect failures instead of masking them. It can be useful for systems which allow a downtime but need to implement fail-silent behaviour or it can also be used as a failure detection mechanism that triggers scrubbing on-demand. Johnson et al. investigated DWC in detail [65]. By means of fault injection experiments and radiation tests, the authors found that

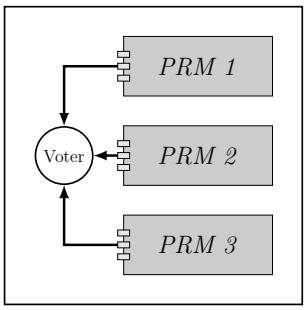

Figure 9: A system utilising dynamic partial reconfiguration with three redundant reconfigurable modules (PRM) and a voter in the static area

DWC can detect approximately 99.85% of all failures at the cost of ca. 200% hardware overhead.

An extension to DWC is proposed by Anderson et al. in [66]. The authors use DWC as failure masking technique by taking advantage of the probabilistic distribution of a circuit's output. The authors use a system comprising five cascaded half-band filters whose output is characterised by a distinct, non-uniform distribution. Once the comparator detects a mismatch, it selects the output with the higher probability by checking a stored histogram. Due to the discrete bins of the histogram, correct detection percentage can be bad for lower significant bits. Therefore, the authors combine this approach with an additional history buffer filled with the last decisions.

Instead of applying spatial redundancy to the netlist of a circuit, the whole circuit can also be seen as a module which is then duplicated or triplicated. This approach is easy to implement [67] but lacks the automatic re-synchronisation after repair which can be achieved by netlist approaches like XTMR. However, a benefit of modular redundancy is the physical separation of the modules which allows a partial (on-demand) scrubbing [44].

Today's usage of modular redundancy is often driven by systems that utilise dynamic partial reconfiguration and in which the design is broken up into physically separated partitions anyway. Thus, it is no surprise that the earlier mentioned systems proposed by Paulsson [40], Jacobs [42] and Straka [68] are all based on modular redundancy. All these systems have in common that one or more voters and/or comparators are placed in the static area, similar to the scheme depicted in Figure 9. The failure detection mechanism monitors the output of redundant modules and triggers an on-demand scrubbing process once a failure has been detected. An interesting as-

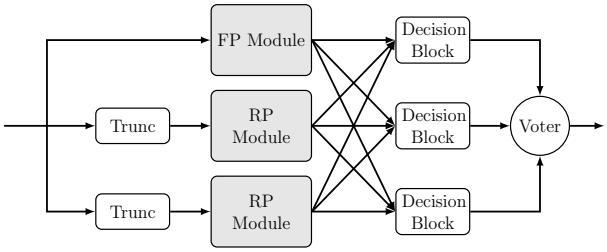

Figure 10: Basic principle of Reduced Precision Redundancy with one full-precision (FP) module and two reduced-precision (RP) modules

pect of such a system is its adaptability as pointed out by Jacobs et al. in [42]: Since redundant modules can be added and removed on-demand, the system availability can be tuned according to external constraints in terms of area and power overhead.

Another technique that can be understood as a modification of modular TMR is RPR. The idea of applying RPR to FPGAs in space systems goes back to several works at the U.S. Naval Postgraduate School, Monterey [69–72] and was later followed up by Bratt et al. [73, 74]. Instead of using three redundant copies of a module, one module processes data with full precision while the two other modules process the data with reduced precision. Hence, RPR is suitable for algorithms that process data which is represented by a block of bits ordered in increasing or decreasing importance, e.g. fixed-point numerical problems [70]. A decision block determines if a failure has occurred as follows [73]:

```
if((|FP_Out − RP1_Out| > T_h) and (RP1_Out = RP2_Out))
then output ⇐ RP2_Out else output ⇐ FP_Out
end if
```

The full precision output FP_{Out} is always chosen if no failure has been detected or if the reduced precision modules disagree. The decision further depends on a threshold level T_h: The full precision module is assumed to be correct if its output differs less than T_h from the reduced precision module output $RP1_{Out}$. For an FIR filter design, Bratt et al. showed in [73] that the failure rate can be improved by ca. 200 times compared to an unprotected design at the cost of ca. 70% hardware overhead. For the same circuit, a full TMR mitigation approach improves the failure rate by ca. 1200 times at the cost of 208% hardware overhead.

2.4.2.2 *Information Redundancy*

Although information redundancy techniques are mainly applied to memory and communication channels, several circuits can profit from them too. Information redundancy techniques add redundant bits to data to be able to detect or even correct falsified information. An example for the first case is the CRC code whereas error correction can be achieved for instance by Hamming codes.

EDAC techniques are often applied to state machines. The states can be encoded using different coding schemes, e.g. binary, one-hot or Gray. In addition, parity bits can be added to achieve a Hamming code which enables the detection or correction of bit upsets. In [75], the robustness of state machines with binary, one-hot, Hamming with a distance of 2 (H2) and Hamming with a distance of 3 (H3) codes was tested using synchronous fault injection. According to the authors, H3 encoding can fully handle single errors and is least affected by double bit errors. State machines with H2 encoding have less overall errors than state machines with one-hot encoding and about half the error rate of state machines with binary encoding. Due to the hardware overhead and the decreased performance, the author concludes that H2 encoding is the best compromise in terms of size, speed and fault-tolerance. For SRAM-based FPGAs, however, these results should be regarded with care because the influence of configuration memory upsets is not taken into account. Using fault injection, Morgan et al. found in [76] that the additional required logic can "potentially add more unreliability than the reliability it adds to the original circuit". An actual implementation of a fault-tolerant state machine that uses Hamming codes is described for instance in [77].

An interesting application of information redundancy for the sake of error detection and correction is ABFT which goes back to the work of Huang and Abraham in 1984 [78]. ABFT is used to implement fault tolerant matrix operations. Recently, Jacobs et al. investigated in [79] the overhead and reliability of ABFT in FPGA systems. The authors use a Multiply and Accumulate (MAC) unit where the inputs are fed from Block RAM and where the output data is written back to Block RAM. One of the implementations uses a second MAC unit that generates and validates the checksums. Compared to TMR, the hardware overhead is as follows: 21% LUT overhead (TMR: 148%), 24% flip-flop overhead (TMR: 84%), 0% Block RAM overhead (TMR: 200%) and 25% DSP48 overhead (TMR: 200%). From 100,000 injected

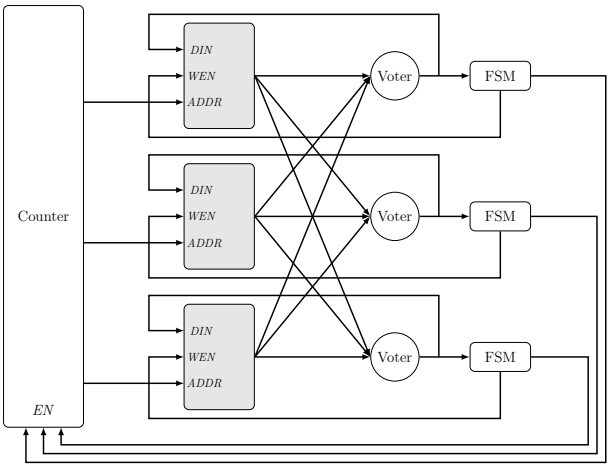

Figure 11: Simplified diagram of a Block RAM to which TMR and scrubbing is applied

faults, 1,216 errors occurred in the unprotected design, 351 errors in the ABFT design and 42 errors in the TMR design.

2.4.3 *Block RAM*

The embedded RAM blocks in Virtex devices need special care regarding the mitigation because very similar to the configuration memory, upsets in Block RAMs can accumulate, leading to an ever decreasing reliability of the memory. Although the Block RAM content can be read out via the configuration interface, external scrubbing is not possible during operation because the RAM can not be accessed by configuration and user logic at the same time [32].

The recommended mitigation approach by Xilinx [80] includes TMR for the Block RAM (including triplicated voter) and a memory scrubbing engine implemented on user logic layer, similar to the scheme depicted in Figure 11. This method can only be applied to single-port RAMs because eventually, they must be replaced by dual-port counterparts as the second port is required for scrubbing. A counter auto-increments the address of the second port and once the voter detects a failure, the counter is stopped, the voted and thus corrected output is written back to memory and the counter is started again.

Rollins et al. present a comprehensive comparison of fault-tolerant memories in SRAM-based FPGAs in [81]. The study covers the TMR and scrubbing approach as described above plus several information redundancy techniques, including duplication with error detection codes and error detection and correction, applied in different config-

urations (with and without triplicating the logic, with and without memory scrubbing). The fault-injection results are similar to what Morgan observed when applying information redundancy to state machines [76]: First, the area overhead of the information redundancy techniques sometimes exceeds the TMR approach and secondly, the failure rates are always even worse than the rate for the unprotected design. Only if Block RAM is used as Read Only Memory (ROM), some of the information redundancy techniques perform slightly better than the unprotected design but always worse than TMR.

2.5 MITIGATION DESIGN TECHNIQUES AIMED AT DESIGN-TIME FAULT AVOIDANCE

Another group of mitigation techniques, which can best be described as fault avoidance techniques applicable during design time are discussed in this section. This group includes analytic approaches that are aimed at analysing the sensitivity of circuits but also at reducing this sensitivity, for instance, by re-routing the circuit design.

As already mentioned in Section 2.4.2, researchers at Politecnico di Torino found that the TMR approach can be invalidated by SEUs because a single configuration bit upset can lead to multiple failures on the user logic layer. By observing and analysing this fault mechanism, a set of tools has been developed which allow the avoidance of these faults already at design time.

In [82], a Static Analyzer tool (STAR) is presented. Based on the researcher's knowledge about the proprietary bitstream format, the tool is able to determine the critical configuration bits of a circuit design. If TMR is applied to this circuit, the tool also determines the bits which can invalidate the TMR approach as described before.

Based on STAR, a Reliability-Oriented Place and Route (RoRA) algorithm has been developed [83, 84]. By re-routing the circuit, RoRA can avoid the problem of single upsets attacking the TMR approach. The authors were able to demonstrate that RoRA minimises the number of wrong answers of circuitry to which TMR or XTMR is applied drastically. Furthermore, the authors point out that RoRA can identify critical configuration bits much faster than fault injection experiments. However, compared to the original TMR version, the re-routing decreases the performance of the circuit.

To increase the performance of circuits to which TMR is applied, a tool called V-Place was then presented in [85] and it was shown that this tool can optimise the circuit's frequency up to 44%.

The STAR tool was later updated to STAR-LX. The main advantages are the reduction of the analysis time of more than five times as well as the ability to analyse the dynamic evaluation of the design under the presence of SEUs [86]. A modification which can analyse the effects of MBUs called STAR-MCU is presented in [87]. To mitigate the effects of MBUs, a new placement algorithm called PHAM was presented in [88].

For engineers and scientists who are interested in building their own netlist analysis and CAD tools, researchers from Brigham Young University present an interesting JAVA toolkit called RapidSmith [89]. The toolkit offers a rich Application Programming Interface (API) to parse, analyse and manipulate XDL files (which can easily be created from Xilinx netlists). A very recent work that makes use of this toolkit is presented in [90]. In this paper, the authors estimate the susceptibility of an FPGA design. To determine the number of sensitive bits that are responsible for the different SEU induced effects, as discussed in Section 2.4.2.1 and shown in Figure 8, the authors conduct the post-routing analysis using appropriate API functions of RapidSmith.

2.6 SIMULATION, EMULATION AND ANALYSIS OF SINGLE EVENT EFFECTS

This section gives a brief overview of techniques for simulation, emulation and analysis of single event effects, including accelerated radiation testing and fault injection. These techniques are necessary to validate any mitigation methodology applied to the design.

2.6.1 *Accelerated Radiation Testing*

Although first in-flight data for Virtex-4 devices have been published [25], the common way to gain reliable static and dynamic cross-sections for these devices is by means of accelerated radiation testing.

To simulate high-energy galactic cosmic rays and solar event heavy ions on ground, the FPGA is exposed to low energy ions available in particle accelerators. The quality of the simulation can be evaluated by the amount of energy lost per unit length of track, also referred to as Linear Energy Transfer (LET). Because the SEE sensitive region

is rather thin, ions with lower energies are sufficient for simulation as long as the LET is similar to the one of galactic cosmic rays and solar event heavy ions. The typical energy range used for simulation is of the order of several $\mathrm{MeV/A}$ and the penetration range is between 30 and 100 µm. In general, the estimation of the SEU sensitivity using this concept is rather conservative [91]. The machine most commonly used for heavy ion SEU testing is the cyclotron. Several accelerators can be found across Europe, for instance the facilities GANIL and IPN in France, SIRAD and LNS in Italy, GSI in Germany and the HIF in Belgium [92].

Single event phenomena can also be induced by protons. Linear accelerators and cyclotron accelerators are capable of generating protons with sufficient energy to simulate solar flare and proton belt conditions [15].

Regarding Virtex devices, most test result data has been collected at the cyclotron at Texas A&M University and/or at the cyclotron at Lawrence Berkeley National Laboratory and published by Los Alamos National Laboratory, NASA Goddard Space Flight Center and the Xilinx Radiation Test Consortium. Quinn et al. present in [93] results regarding radiation-induced MBUs in Virtex, Virtex-II and Virtex-4 devices and discuss the general reliability concerns of Virtex FPGAs in [94]. In [60] they present the results regarding circuitry to which TMR is applied and discuss the problem of domain crossing errors. In [95], first results for Virtex-5 are published. In 2009, results regarding the SEU-susceptibility of logical constants were presented [96]. In the same year, a paper describing their methodology for static and dynamic testing was published too [97]. The upset characterisation of embedded PowerPC cores is presented by Allen in [98] and the more general characterisation for Virtex-4QV FPGAs by Swift in [99], finally leading to the summary report published by NASA and Xilinx [3]. One year later, a report summarising the results gathered from dynamic testing and from the testing of protected designs was published by Allen [24]. Recently, the static SEU characterisation of Virtex-5QV was presented in [23].

2.6.2 *Fault Injection*

As an alternative to accelerated radiation testing, upsets in the configuration memory can also be emulated by fault injection.

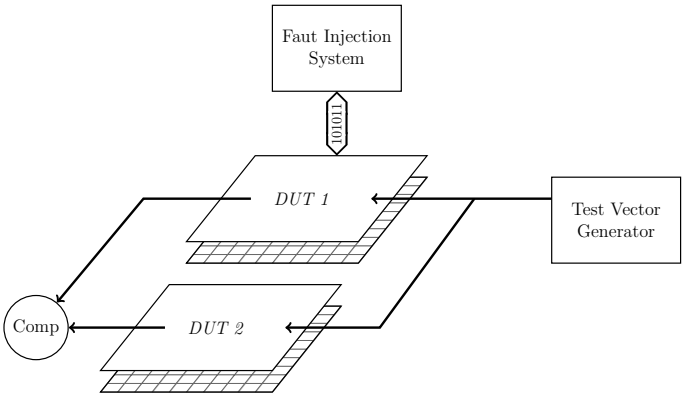

Figure 12: Example of a fault injection system

Although many different fault injection implementations have been presented in literature, the basic structure is similar for many systems, see Figure 12. Two devices are simultaneously fed by test vectors. One of the devices is used as a 'golden' reference while faults are injected into the second Device Under Test (DUT). Fault injection is based on bitstream manipulation: Often, a configuration frame is read back from the FPGA via one of the configuration interfaces, one or more bits are flipped and the frame is written back to the device. The outputs of both FPGAs are compared by some mechanism which detects if the fault injection led to a failure. Alternatively, only one DUT can be used and its response is compared to 'golden' answers during the fault injection campaign. In the following, two exemplary European systems are presented.

A fault injection system with a long history in Europe is FLIPPER, developed by Alderighi et al. under ESA funding [100]. The system comprises one Virtex-II Pro FPGA as controller that can be connected to a DUT board hosting the FPGA under test. The controller communicates with a software running on a host PC via USB. In contrast to the example depicted in Figure 12, only one DUT is used and its response is compared to stored, known to be correct answers. Test vectors can be converted from testbench stimuli and are fed into the DUT after one or more faults were injected into the bitstream. Results from FLIPPER were compared to acceleration testing results [101] and the authors conclude that FLIPPER is an effective tool to evaluate different mitigation techniques due to its capability to predict failure rates, provided that raw configuration bit upset rates of the target environment are known. However, the authors also point out that the failure rate might be underestimated because SEUs in flip-flops, SETs and MBUs are not emulated. FLIPPER was also compared to and

used for the analysis of the STAR/RoRA tool as described in Section 2.5 [102].

The second fault injection system developed under ESA funding is FT-UNSHADES from University of Seville. Compared to FLIPPER, its initial aim was the emulation of SEUs and MBUs that originate from SETs and thus manifest in flip-flop cells rather than the emulation of configuration memory upsets. The system is based on a Virtex-II and the circuit under test is duplicated and compared within one FPGA. In each campaign, the application is driven to the desired fault injection time, the clock is stopped, the fault is injected into the desired flip-flop(s) of one of the circuits, the clock is restarted and the outputs of both circuits are compared to detect any mismatch [103, 104]. Later, the injection system was updated (FT-UNSHADES-uP) to allow a more in-depth analysis of microcontrollers. The system only uses one circuit under test whose output is compared to the theoretically correct output and it is now able to also inject faults into Block RAMs, LUTs and SRL16s [105, 106]. More recently, FT-UNSHADES2 has been developed [107]. The system is based on Virtex-5, and all data management is processed in hardware, leading to much higher fault injection rates. Now, the system can also be used to inject faults into the configuration memory and the usability was increased due to a simplified design flow and a web browser based user interface.

2.6.3 *Availability Analysis*

Regarding availability analysis methods for SRAM-based FPGAs, only a small amount of work exists. McMurtrey et al. use Markov models to estimate the reliability of TMR systems [53]. Among other things, the authors investigate how multiple TMR partitions increase the reliability. Ostler et al. present a reliability analysis of SRAM-based FPGAs in [108]. The methodology takes particular radiation environments into account and is based on fault injection experiments. Kastil et al. present a dependability analysis of their fault tolerant systems in [109]. Applications hosted on SRAM-based FPGAs are partitioned into functional units to which different redundancy configurations can be applied. An automatic tool creates a Markov model of the overall system. Martin et al. also uses Markov chains in [110] to model the availability that can be achieved with different scrubber implementations. In two recent publications, Hoque et al. deal with probabilistic model checking techniques for aerospace applications [111, 112]. The

authors use a continuous-time Markov reward model to determine the availability of the configuration memory when scrubbing is applied as recovery technique.

2.7 RESEARCH PLATFORMS FOR SRAM-BASED FPGAS IN SPACE

This section presents an overview of several research platforms that comprise SRAM-based FPGAs. It summarises concepts, which could possibly be applicable to future spacecraft data processing systems.

Flight heritage for Virtex-4 and Virtex-5 is relatively rare and many of the payloads serve only as technology demonstrators so far. From the publicly available information, it seems that none of the platforms utilises dynamic partial reconfiguration as a functional feature. Although dynamic partial reconfiguration may offer benefits for some projects, it can be assumed that in-flight experience for this unique capability of SRAM-based FPGAs is still a long way off.

However, in the research community, this topic is actively investigated. In the following, six platforms and frameworks are presented that comprise Virtex devices which utilise dynamic partial reconfiguration and which specifically target space applications. A summary of the systems is given in Table 7.

The platforms can coarsely be classified by the way the reconfigurable modules are used. Most of the platforms implement a System on Chip (SoC) in which the reconfigurable modules are connected to a soft Central Processing Unit (CPU) core, i.e. the reconfigurable modules are used as hardware accelerators that can be installed on demand. The other group of platforms uses reconfigurable modules as processors that can process data streams independently and thus without interaction of a CPU.

2.7.1 *Reconfigurable System on Chip*

A demonstrator platform called Dynamically Reconfigurable Processing Module (DRPM) is under development at University of Bielefeld and Paderborn, Germany [113]. It is based on a prototype platform called RAPTOR-X64. Aside from a communication module, the system comprises two processing modules, each module including a Virtex-4 FPGA. The reconfiguration controller is part of the FPGA. It is not only used to reconfigure the FPGA via the ICAP interface but also for scrubbing of the configuration memory. The partial recon-

Table 7: Comparison: Research platforms for SRAM-based FPGAs in space

System Architecture	
Braunschweig	Macro-Pipeline Multiprocessor
Brno	Hardwired hardware blocks
Bielefeld/Paderborn	General Purpose SoC
Edinburgh/IKERLAN	Reconfigurable Computer
Florida	General Purpose SoC
Arizona	Several FPGAs hosting single task processors
Communication Mechanism	
Braunschweig	Network on Chip
Brno	Hardwired
Bielefeld/Paderborn	Embedded Macro Bus Structure
Edinburgh/IKERLAN	via ICAP
Florida	PLB Bus
Arizona	OPB Bus
Key Features	
Braunschweig	NoC used to isolate modules during the reconfiguration process.
Brno	Reconfiguration controller implemented in hardware.
Bielefeld/Paderborn	Embedded Macro to allow flexible placement of modules.
Edinburgh/IKERLAN	Modules are handled as hardware tasks. Communication via ICAP to allow flexible placement of modules.
Florida	Adaptive fault-tolerance.
Arizona	Two-level healing methodology.
FDIR Strategy	
Braunschweig	Scrubbing.
Brno	Different redundancy modes and on-demand scrubbing.
Bielefeld/Paderborn	Scrubbing.
Edinburgh/IKERLAN	Scrubbing due to continuous task reconfiguration. Fault-aware task allocator (hard errors). Fault-tolerant ICAP controller.
Florida	Adaptable, modular redundancy and on-demand scrubbing.
Arizona	Software-based fault detection. Partial on-demand scrubbing. Cold module redundancy. Task allocation to another FPGA.

figurable modules are connected using so-called Embedded Macros which embed a bus structure into tiles. The main motivation for such a structure is given in [114]: By dividing a partial reconfigurable area into atomic units called tiles, modules of different sizes can be more efficiently placed at run-time. It was found that an embedded bus structure with shared signals supports the flexible placement of the modules optimally due to its homogeneity. The tool STARECS from Politecnico di Torino [115] is used to analyse the SEU effects on the system and at present, work regarding the fault-tolerant communication via the Embedded Macros is in progress.

Researchers at University of Edinburgh, UK are also working towards a partial reconfigurable system on Virtex-4 FPGAs. The main objective of the work is a Reliable Reconfigurable Real-Time Operating System (R3TOS), introduced in [116]. Reconfiguration is done by R3TOS internally through the ICAP port. In the course of the research on R3TOS, an Area-Time Response Balancing Algorithm (ATB) for scheduling real-time hardware tasks was proposed in [117] and a task allocator in [118], which is able to deal with spontaneously occurring faults. The paradigm followed in Edinburgh is that hardware tasks are handled like normal threads in a higher programming language (e.g. POSIX threads). To 'call' a hardware task, the reconfigurable module needs an appropriate interface, which is proposed in [119]. In the same paper, an interesting approach for inter-task communication is presented: Instead of utilising a network with a high resource overhead, the data is simply copied from the output buffer of one module to the input buffer of another module by reading the data through ICAP and writing it back. One benefit of avoiding an on-chip communication is the fact that less wires need to cross the partial reconfigurable modules which increases the flexibility regarding the module placement. Built on the ICAP-based communication, a second task allocator called Snake is presented in [120] and because R3TOS is heavily making use of the ICAP port, a fault-tolerant ICAP controller is introduced in [52].

Researchers at University of Florida, USA are working towards a framework for the usage of Commercial Off-The-Shelf (COTS) FPGAs in space applications. The system also utilises dynamic partial reconfiguration and one main aspect of their work is adaptable fault-tolerance that is achieved by adding and removing redundant reconfigurable modules depending on external constraints in terms of availability and power [42, 121]. The basic structure of the proposed

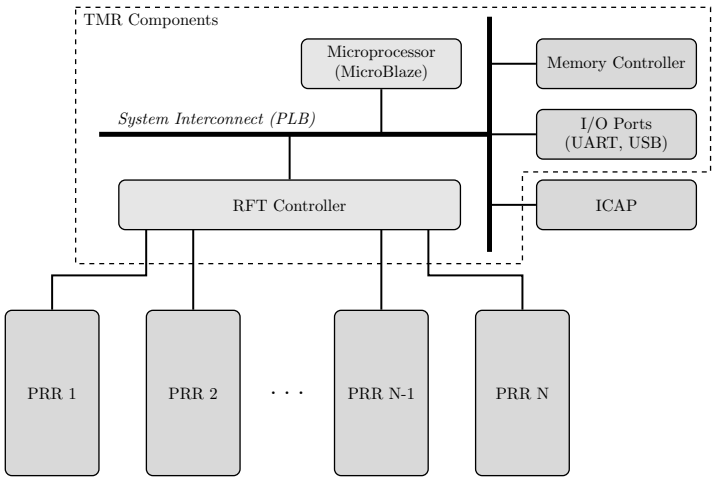

Figure 13: Framework as proposed by Jacobs et al. (University of Florida), redrawn from [42] ©ACM, 2012

framework is depicted in Figure 13. Several reconfigurable partitions are connected to a controller that comprises failure detectors. The controller itself is connected via a PLB bus to an on-chip Microblaze softcore. The controller, the bus and the softcore, as well as its peripherals, are placed in the static area of the FPGA. The static area is hardened against SEUs by applying TMR to the netlist of the design. The researchers also investigated the suitability of ABFT for such systems [79] and presented their own fault injection system [122].

Researchers at University of Arizona, USA, are working on a Virtex-5 based partial reconfigurable system called SCARS which is introduced in [41] and based on a two-level healing methodology. The system comprises five Virtex-5 FPGAs, each including a Microblaze soft core which is responsible for the self-healing. The partial reconfigurable modules are partitioned together with redundant copies into so-called slots and connected to the Microblaze processor bus. The software running on the Microblaze is responsible for the detection of faulty modules. If a fault is detected, the module is scrubbed through the ICAP interface. If the failure persists, it is seen as an hard error and another redundant module in the slot is activated. The five FPGAs are connected to a master node in a wireless network. Once all modules in a slot are faulty, the task which was running in the faulty slot is moved by the master node to another FPGA.

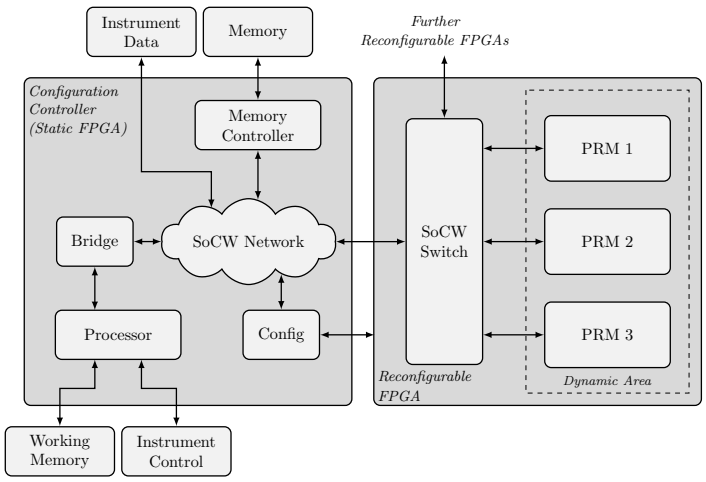

Figure 14: DRPM by TU Braunschweig/Astrium Ltd., redrawn from [123] ©IEEE, 2011

2.7.2 *Reconfigurable Stream Processors*

The work on reconfigurable FPGAs at the University of Braunschweig, Germany, goes back to a data processing unit for a camera on-board the Venus Express mission. In 2007, an update of this architecture was proposed which also allows in-flight partial reconfiguration [124]. To interconnect the partial reconfigurable modules, a Network on Chip (NoC) called SoCWire was proposed in [125] which is heavily based on SpaceWire, the only space-qualified point-to-point network architecture [126]. The main motivation to use a NoC approach can be found in [127]: During the reconfiguration process, glitches can occur since frames become directly active during the write cycle. To qualify such a system for space applications, the partial reconfigurable module must be isolated from the host system which can optimally be achieved by a NoC approach. The successful isolation and thus protection against such glitches was shown in [128]. SoCWire became part of a demonstrator platform called DRPM, developed in cooperation with the European Space Agency and Astrium Ltd., UK. The demonstrator comprises one or more modules, each module with a radiation hardened reconfiguration controller and two Virtex-4 devices. In 2011, an Advanced Microcontroller Bus Architecture (AMBA) to SoCWire bridge was presented in [123] and recently a higher protocol called SoCP, which is also based on a SpaceWire protocol, was introduced in [129]. The basic structure of the DRPM can be seen in Figure 14. The reconfiguration controller, depicted on the left hand

side of Figure 14, is implemented on a reliable antifuse FPGA. It comprises a SoC with a LEON3 CPU and several peripherals, e.g. memory controllers. The Virtex-4 FPGAs, one of them depicted on the right hand side, are divided into reconfigurable partitions which are interconnected via a SoCWire routing switch.

A more theoretical framework that deals with FDIR for SRAM-based FPGAs is proposed by researchers at University of Brno, Czech Republic. Several hardware blocks are arranged in a hardwired processing pipeline. For each hardware block, a different redundancy mode can be applied, e.g. TMR or DWC. The output of the failure detectors is connected to a bus and the health status of the hardware blocks is reported via this bus to a reconfiguration controller. In contrast to other approaches presented here, the reconfiguration controller is implemented in hardware. In the course of this research, the design of online failure checkers was first proposed in [130] and later extended to the overall framework [68]. The reconfiguration controller is described in [43] and a fault injection system is presented in [131]. Finally, a dependability analysis for the framework is described in [109].

2.8 SUMMARY OF EXISTING MITIGATION TECHNIQUES

Research on mitigation techniques for SRAM-based FPGAs for space applications has engendered a very large number of publications in this research field. As it was shown in Section 2.3.3 the proposed methodologies can be split into just a few main categories targeting (i) the configuration memory, (ii) the user logic or (iii) the Block RAMs. Regrettably there are not enough design details in the open literature in order to compare the existing methods fairly. In addition, onboard designs are very much dependent on mission objectives and constraints. In most cases, the decision on the use of a particular technique will be based on a trade-off between power, area and performance overheads as well as achievable system availability. In this section a summary of the reviewed mitigation methods is presented which is illustrated by an example decision strategy on selecting the right mitigation technique. It is hoped that the proposed decision strategy can serve as a guidance to researchers and engineers who are novices in the field. However, it is expected that designers will exercise their own judgment and draw their own conclusions taking

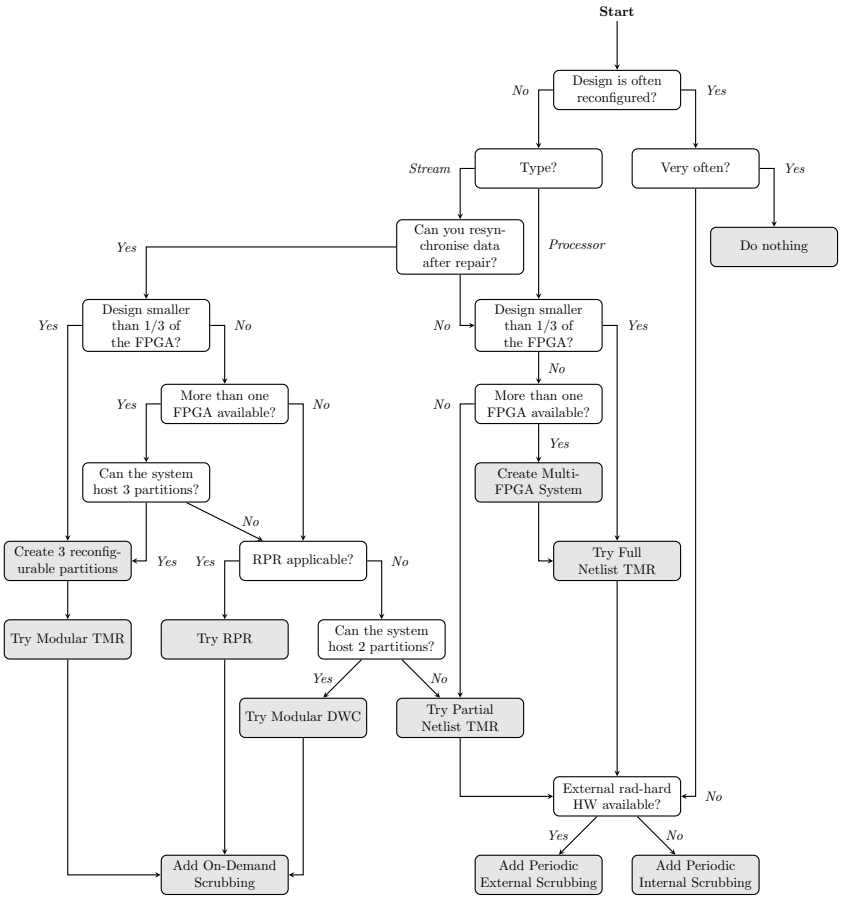

Figure 15: Example decision flow

into account the specifics of their projects when considering the recommendations given below.

Figure 15 exemplifies the main steps, which a decision process on selecting a particular mitigation technique for SRAM-based FPGAs on board spacecraft might involve. Considering the fact that space engineering projects usually require strict verification procedures, it might be a wise decision to choose a solution with the lowest possible implementation complexity to meet the given constraints.

If the FPGA design is often reconfigured, for instance because the chip area is shared by several applications, it could be decided not to apply any mitigation technique at all. This is because every time the system is reconfigured it is brought back into a safe initial state, i.e. possible faults in configuration memory and user memory are removed too. This simple solution has no additional power, area or performance overhead at all. If it does not lead to a satisfactory system availability, however, one may add periodic scrubbing which is able

to remove faults in the configuration memory during the time the system is running. Still, failures can be 'trapped' in state-dependent user logic but fortunately, the ratio of user memory elements to sensitive configuration bits is often small enough to gain a significant increase in system availability anyway.

If frequent full reconfigurations are not part of the normal operation, one must consider to add a combination of mitigation techniques. Applying spatial redundancy without implementing a repair strategy (like scrubbing) is not recommended because it only extends the time span until the system becomes unreliable. On the other hand, a strategy solely based on scrubbing is not an ideal solution either because faults can manifest as permanent failures within the user memory logic. Thus, a failure detection and/or failure masking technique is usually combined with a failure recovery technique.

Most payload data and imaging applications could be classified as either being of a *processor* type or a *stream* type. The first type comprises all kinds of microcontrollers and -processors or custom-built processors used for data acquisition and similar tasks. These types of applications are never or rarely reset or restarted and may contain a large state space and a huge amount of state variables. Embedded RAM is often used to store data over a long period of time. The second type comprises circuits that can mainly be found in payload data processing applications, for tasks like data compression, encryption or filtering. These types of applications process data block-wise, e.g. image by image, and often, the state space is traversed with each data block. Typically, the number of state variables is low and embedded RAM is mainly used for FIFO buffers.

Two possible mitigation strategies can be followed:

1. Spatial redundancy (partial TMR or full TMR) is applied to the netlist of the circuit and the configuration memory is periodically scrubbed.

2. The whole circuit is duplicated or triplicated (modular redundancy) and a majority voter or comparator, respectively, is used as failure detector. Then, the failure recovery can be triggered on demand.

The spatial redundancy mitigation strategy goes well with the *processor* type of application because this low-level redundancy approach allows automatic data resynchronisation after repair. This strategy

is also simple to apply because commercial tools exist which automate the insertion. However, one must carefully verify that the used toolchain does not optimise the inserted redundancy away. If the design is small enough and the power budget relaxed, full TMR leads to the best possible system availability. If the design is too large to apply full TMR, one could either use a multi-FPGA system or apply partial TMR. With partial TMR, only the feedback loops are typically protected but not the data path within the user logic. As a consequence, errors will become visible as transient failures at the output of the FPGA. No matter which kind of TMR is applied, one must implement periodic scrubbing too. If external radiation-hardened hardware is available or can be afforded, external scrubbing is the more reliable approach. If either blind or readback scrubbing should be used depends on the particular application but blind scrubbing is surely the solution with the lowest implementation complexity.

The modular redundancy mitigation strategy goes well with the *stream* type of application because the user logic can be brought back to a safe initial state after each data block. One drawback of this mitigation approach is the fact that state variables must be synchronised between redundant instances after repair. But since data is processed block-wise and the number of state variables is usually low, technical solutions can be found to execute the data resynchronisation between the data blocks. Despite the increased implementation complexity, this mitigation approach offers some real benefits. For instance, the configuration memory must only be repaired after a failure has been detected, which can maximise the system availability and minimise the power consumption compared to the periodic scrubbing approach. Often, the *stream* type of application does not require maximum possible system availability. For instance, one may tolerate a short downtime with each upset as long as the system is fail-silent. In this case, it is sufficient to only duplicate the circuit and to use a comparator as failure detector. If downtime is not an option, modular TMR can be applied. If not enough chip area is available to triplicate the circuit, one can investigate if RPR is applicable. Then, failures can be masked but the output of the circuit might be degraded in precision until the faulty module is repaired. Modular redundancy goes also well with multi-FPGA systems because redundant instances can be distributed over several FPGAs.

2.9 ANALYSIS OF THE STATE OF THE ART

Research regarding mitigation techniques for SRAM-based FPGAs can coarsely be broken down into:

- Work on research and demonstrator platforms, which may also include one or more of the topics mentioned hereafter.
- Work on specific failure detection and masking techniques (TMR, DWC etc.).
- Work on specific failure recovery techniques (mainly scrubbing).
- Work on fault emulation techniques.
- Work on fault avoidance techniques by improving the design tools.
- Work on dependability analysis techniques.

Both, University of Braunschweig and University of Bielefeld provide hardware platforms which are scalable and which already implement features very similar to real flight hardware. However, the platforms do not implement any FDIR scheme yet. Due to the support from the European Space Agency and Airbus Defence and Space, both platforms are available for a proof of concept implementation of the FDIR methodology developed in the course of this PhD work. Although both systems offer unique capabilities, the one by University of Braunschweig is chosen for this work because of the following reasons:

- The system designed by University of Bielefeld is a SoC with an embedded softcore on each FPGA whereas the system from University of Braunschweig is based on stream processors that can process incoming streams without the interaction of a CPU. The FDIR methodology proposed in the course of this PhD work aims at high performance computing which can best be achieved with a system in which the algorithms are solely implemented in hardware. Therefore, the proposed FDIR strategy concentrates on such systems rather than on SoCs and as a result of this, the system from Braunschweig is the preferable choice.
- The system from University of Bielefeld includes an internal reconfiguration controller which uses the internal ICAP interface

whereas the system from Braunschweig includes an external, reliable reconfiguration controller which configures the FPGA via the SelectMAP interface. As discussed in Section 2.4.1.4, external reconfiguration and scrubbing is commonly seen as the more robust approach.

- The communication within the system from University of Braunschweig is based on a NoC whereas the system from Bielefeld relies on a custom designed embedded bus structure. For the here proposed approach, a NoC is the preferred choice. Due to the nature of a NoC, the system is scalable, i.e. the available bandwidth is not decreased if the number of network nodes is increased. As a result, a NoC is a sound backbone for high performance computing.

Two research platforms that are intended for the research on FDIR are presented by researchers at University of Florida and University of Brno. Although both platforms offer some unique and original features, e.g. a reconfiguration controller fully implemented in hardware (Brno) or the possibility to add and remove redundant modules during flight (Florida), they are both lacking the flexibility and scalability needed for applications like the ones targeted by the FDIR methodology proposed here. Both approaches are restricted to one FPGA and thus, the FDIR scheme cannot be applied to multi-FPGA systems. Furthermore, the approach of University of Florida is a SoC which may not fulfill the needs for high performance computing. A platform that targets multi-FPGA systems is proposed by researchers at University of Arizona. In this approach each FPGA implements a SoC, in which failure detectors, implemented in software, detect faulty hardware modules. The approach can best be compared to a distributed computing system, since hardware tasks are moved to other network nodes in case of permanent failures. Although this approach is interesting due to its two-level FDIR methodology, it mainly takes hard errors into account and it can be assumed that it also would not fulfill the performance requirements needed here.

Regarding failure detection and failure masking techniques, two approaches could be followed. The first approach would be based on spatial redundancy at netlist level. Several research groups, e.g. at Brigham Young University and University of Turin, provided valuable contributions to this field. The second approach would be based on modular redundancy as used for example by Paulsson, Jacobs and

Straka. For this work, a framework based on the latter approach is chosen because:

- The approach becomes rather technology and application independent.
- The approach allows the tuning of the system availability and the power consumption during flight.
- The approach is easier to qualify for space applications and simplifies the development of stochastic system availability models.
- The approach allows on-demand scrubbing, i.e. a faulty module is only repaired after a failure has been detected.
- The approach goes well with systems utilising dynamic partial reconfiguration as a functional feature since the reconfigurable partitions are already physically separated.

As discussed in Section 2.6.3, not much work exists regarding availability analysis methods. However, it is believed that such analysis methods are important for every FDIR framework as they can provide statements on the quality of the employed FDIR techniques. Thus, the FDIR framework developed in the course of this work should also include such a method. It was found that previous works take only the configuration memory into account. However, other FPGA building blocks, especially Block RAMs, cannot be ignored due to the large amounts of buffers that are typically used by streaming applications. Thus, a more complete, yet easy to follow availability analysis methodology will be presented in this thesis, which also takes Block RAM and flip-flop upsets into account.

2.10 CONCLUSIONS

The literature survey reveals a large number of high quality publications in the field of SRAM-based FPGAs, including work on failure detection, failure recovery and fault simulation. In the course of this PhD work, elements from all these research fields will be evaluated and assembled to one overall FDIR methodology which will be later, as a proof of concept, implemented on a hardware platform developed at University of Braunschweig, Germany. Other research groups exist that also aim at finding such an overall FDIR methodology. However, none of these groups proposed a solution that is flexible and

scalable enough to allow high performance computing with stream processors that can be distributed over several FPGAs. Summarised, the following gap in the research landscape can be identified:

- No FDIR scheme exists that specifically targets multi-FPGA systems in which stream processors can freely be distributed over several FPGAs.
- No availability analysis method exists, which specifically targets such a FDIR framework and which also takes Block RAM and flip-flop upsets into account.

The undertaken literature survey is the first attempt at presenting a systematic and analytical overview of this rich research field, which has not been subjected to a thorough analysis in recent years. In addition, the proposed design recommendations, resulting from the survey, can facilitate a fast start to the topic for both scientists and engineers, who are novices in the field.

3

CHAPTER 3: DISTRIBUTED FAILURE DETECTION METHOD

3.1 INTRODUCTION

The analysis of the state of the art in Section 2.9 revealed that no FDIR approach for SRAM-based FPGAs exists, which specifically targets multi-FPGA systems. Furthermore, all approaches described in literature so far do not allow a flexible placement of stream processors on reconfigurable partitions because usually, the connections between redundant hardware modules and failure detectors are hardwired.

In this chapter, a novel technique called *Distributed Failure Detection* is proposed that shifts failure detection mechanisms to a network level. Instead of hardwired connections, the stream processors are interconnected via a NoC. This NoC can span across several FPGAs, making this FDIR approach especially applicable to multi-FPGA systems.

This novel method, which can also be understood as an FDIR hardware framework, comprises the following main components:

- Data processing is done by so-called stream processors, which can process incoming network data streams automatically. Each stream processor implements a specific data processing algorithm, e.g. data compression, encryption, filtering etc.
- The stream processors are placed on dynamically reconfigurable partitions implemented on SRAM-based FPGAs. The partitions are interconnected with each other in an arbitrary NoC topology via routing switches.
- If increased system availability is desired, a stream processor can be duplicated or triplicated, i.e. the processor works in modular, hot redundancy.
- Adding or removing redundant processors could be done during operation, i.e. the FDIR method is adaptive to reliability, area and power constraints.

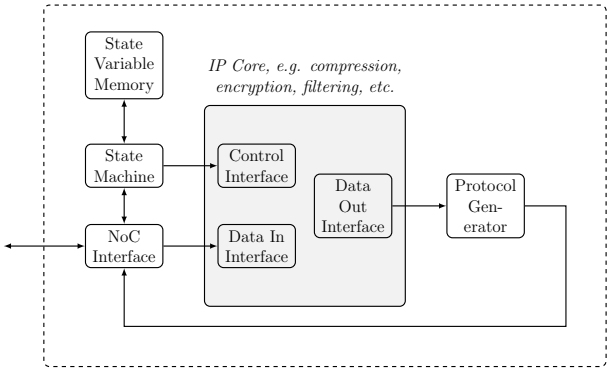

Figure 16: Stream processor architecture

- Failure detection and masking is done by one or several voter modules, which are integrated into the network. A voter module can also work as a comparator. The module outputs a health status, which can be used by an external controller to initiate a failure recovery process.
- If a stream processor is used in hot redundancy, the input network streams must be multicast within the network. Such a multicast mechanism is also part of the FDIR framework.
- A special addressing scheme is required to isolate possible failures propagating through the network to avoid congestion.
- All the aforementioned mechanisms are integrated into a custom-designed NoC routing switch.

The chapter is structured as follows. In Section 3.2, the required architecture of a stream processor is defined. Section 3.3 discusses an example network topology and explains the flow-control mechanism of the NoC. Then, a Failure Mode and Effects Analysis (FMEA) is presented in Section 3.4 that analyses possible failure modes, which can occur in network streams. Section 3.5 presents the core FDIR components, which are designed according to the outcome of the FMEA, including the voter mechanism, the multicast mechanism and the addressing scheme. In Section 3.6, a possible data resynchronisation technique is proposed, which utilises the proposed FDIR components. Finally, Section 3.7 concludes the chapter by summarising the proposed novelties.

3.2 STREAM PROCESSOR ARCHITECTURE

In the proposed FDIR framework, the different processing steps are executed by dedicated *stream processors* which can process incoming data streams independently.

The architecture of a typical FPGA based stream processor is shown in Figure 16. An Intellectual Property (IP) core, the purpose of which is to accelerate a desired DSP functionality, is embedded into the stream processor. The stream processor further comprises a NoC interface for the data exchange, some state machine logic and a memory for state variables. Input control words are interpreted by the state machine whereas input data words are directly fed into the accelerator IP core. An additional memory holds all variables necessary to configure the IP core. If the processing pipeline uses a specific protocol, a protocol parser and/or protocol generator may be added to the inputs and outputs of the core.

The employed IP cores are of a passive nature, e.g. they represent hardware accelerators that are designed to be connected as slaves to a CPU bus. With the additional logic in the stream processor, however, the cores become intelligent enough to process incoming data without the interaction of a CPU, solely by interpreting data and command words in the network input stream. Furthermore, suitable IP cores process input data block-wise. For instance, such a block could be a line of pixels, a full image or a series of images (think of hyperspectral image compression and similar applications). As a rule of thumb, it should be possible to reset the core after each data block without affecting the processing of subsequent blocks.

All FPGA building blocks within one processor are multiplied too if modular redundancy is applied. Therefore, an upset in a user memory cell can manifest itself as a measurable failure in the network output stream in the same way as an upset in the configuration memory. Regarding the Block RAMs of SRAM-based FPGAs, Rollins et al. [81] showed that redundancy is the only effective mitigation approach because other mitigation techniques (like error detection and correction codes) rely on additional logic that must be implemented as part of the potentially unreliable FPGA fabric, see also Section 2.4.3.

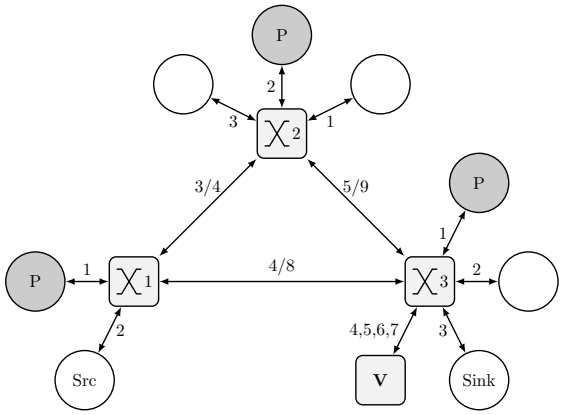

Figure 17: Example network topology

3.3 NETWORK TOPOLOGY

With the Distributed Failure Detection methodology, failure detectors become part of a network. This novel approach allows the free distribution of redundant stream processors throughout the network because the output of each processor can be routed to any failure detector, independent of its location in the network. The network can even span over several FPGAs, i.e. links may connect network nodes on-chip but also off-chip. The possibility to place redundant stream processors on several FPGAs has many advantages, for example in cases where the chip area of one FPGA is not sufficient to host a full fault-tolerant design.

An example network topology is shown in Figure 17. Several partitions (circles) are interconnected via routing switches. A processor has been triplicated and the resulting instances (grey circles) are placed on some of these partitions. Say, data is sent from a source node *Src* to the processor and the processor sends the processed data to sink node Sink. As the processor is triplicated, the data must first be multicast within the routing switches. For instance, routing switch 1 multicasts the packets to output port 1, 3 and 4. In switch 2 and 3, the packets are then routed to the other two redundant instances. After processing, the resulting packets are routed to the failure detector, which in this case is the voter module **V**, connected to routing switch 3. Finally, the output of the voter is routed to the sink node. Typically, the network packets do not arrive simultaneously at the redundant processors. The latency between each redundant processor and the failure detector may differ too. In addition, the partitions might be

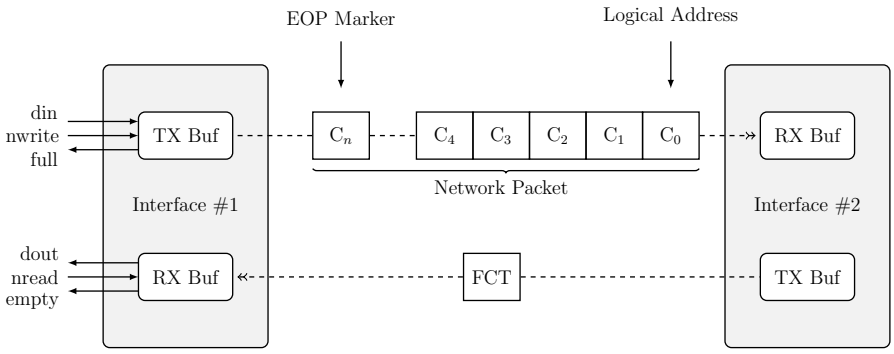

Figure 18: Flow-control mechanism between two nodes

implemented in different clock domains. As a result, the voter module must be able to deal with asynchronous network streams.

To enable a synchronisation of incoming network streams, the NoC must be flow-controlled as shown in Figure 18. Each network packet may start with a logical address (that is typically used for routing within switches) and is terminated by an End of Packet (EOP) marker. Every time the receive buffer has space for eight more characters, the receiving node sends out a Flow Control Token (FCT). Therefore, the receiving node can easily apply back-pressure to the communication channel, i.e. it can force the source node to freeze by ceasing the transmission of further FCTs.

3.4 FAILURE MODE AND EFFECTS ANALYSIS

To draft the voter design, it is first necessary to understand the failure modes that can occur in the network output streams of the stream processors since these must be fully covered by the failure detector.

It was found in an FMEA that two types of failure modes must be expected, those which affect the payload of network packets (application data) and those which affect the network traffic itself. A short summary of the analysis is shown in Figure 19. In detail, the following failure modes were found:

- Case (A): Typical operation. The network packets are identical but may arrive at different points in time at the voter module. This non-synchronicity can be handled by exploiting the flow control of the network architecture. The voter module applies backpressure to the network channels that already received some data until at least one data character has arrived in all slots (i.e. a receive buffer assigned to a particular processor).

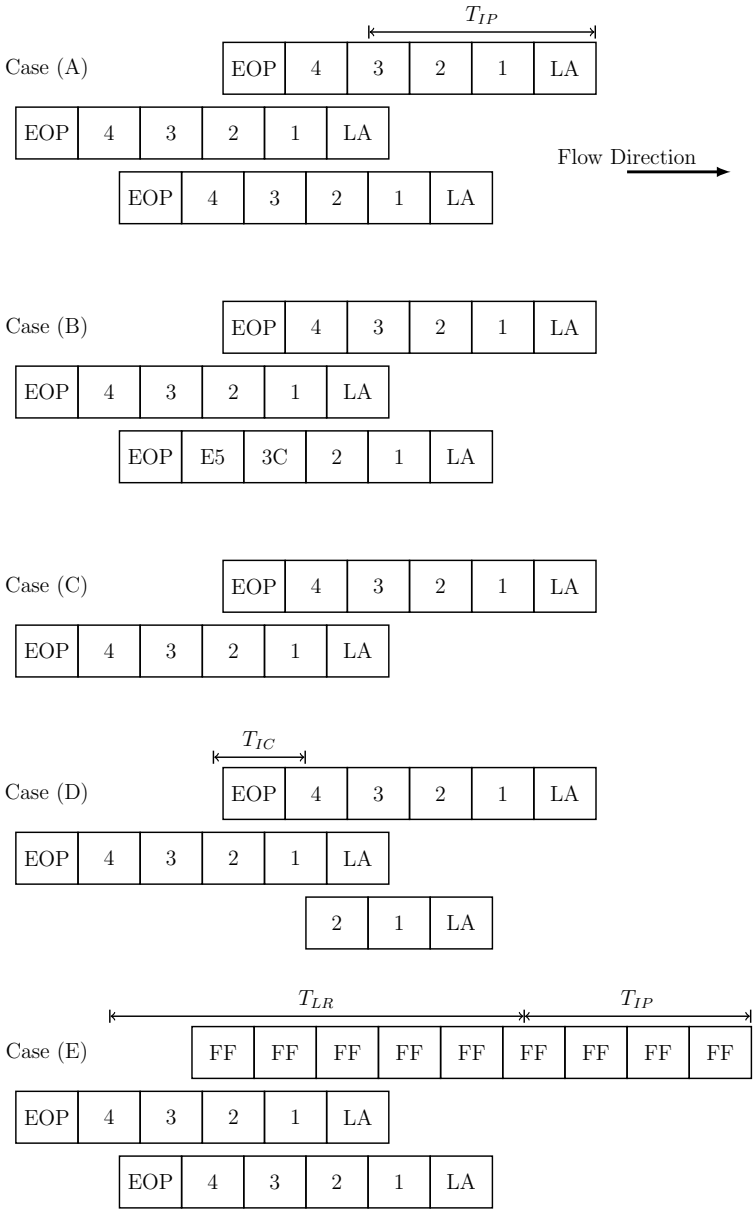

Figure 19: Different failure modes found during FMEA

- Case (B): The network packets have an identical structure, i.e. the network protocol is faultless, but their payload differs due to a failure in the application. In most applications, this case will be the most observed one because the probability of a failure in the application is usually larger than the probability of a failure in the network related components. This failure mode can be detected by a majority voter or comparator mechanism that compares the (synchronised) network streams character by character.

- Case (C): One of the network packets does not arrive at all. It seems as if the corresponding processor became faulty and ceased the transmission for some reason. This case can be handled by a timeout mechanism with a timeout value T_{IP}, hereafter also referred to as inter-packet timeout, which is triggered once the first redundant packet arrives in one of the slots of the voter module. If the timeout elapses and one of the redundant network packets has not arrived in its slot, this slot is marked as faulty.

- Case (D): The transmission of one of the network packets suddenly stops before the EOP marker is reached. This case can be handled by a second timeout mechanism with a timeout value T_{IC}, hereafter also referred to as inter-character timeout, which is always retriggered when data character(s) are available in some slots but not in others. If the timeout expires, it must be assumed that the processor associated with the still empty slot suddenly stopped the transmission and thus this slot is marked as faulty.

- Case (E): One processor becomes a *babbling idiot* and is transmitting undefined data at undefined points in time. Dealing with this case can be problematic if the data from the babbling idiot arrives much earlier than the data from the two other healthy processor instances. Then, the inter-packet timeout would expire first and the two healthy slots would be spuriously marked as faulty (actually all slots would be marked as faulty because further voting is not possible). As it is rather unlikely that two processors fail at the same time, this case is handled by assuming that the early packet is wrong, i.e. the corresponding slot is temporarily marked as faulty. Then, a second timeout value T_{LR}, hereafter also referred to as last-resort timeout, is started. If

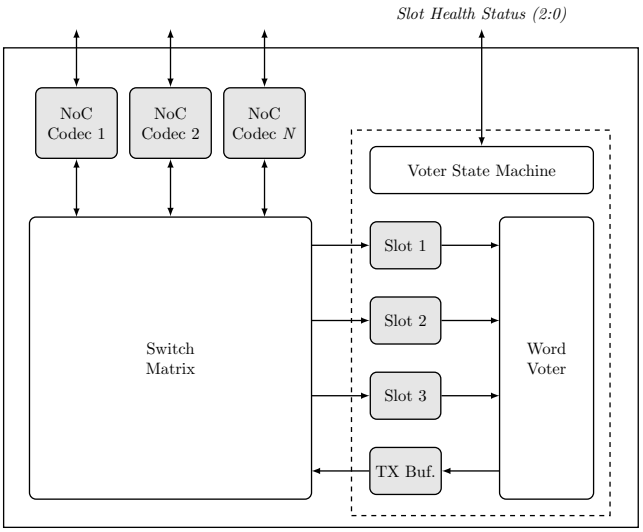

Figure 20: Voter module embedded into NoC routing switch

the packets from the healthy processors arrive within this timeout period, no further action is required. If they do not arrive, however, all slots must be marked as faulty.

Aside from the aforementioned failure modes, another mode must be considered when multicasting data. If a processor becomes faulty, it may block incoming traffic. This case can be handled by using a non-blocking multicast mechanism that comprises a multicast timeout. If one of the processors blocks incoming data throughout the timeout period, it is afterwards excluded from the multicast until the end of the current packet transmission.

3.5 FAILURE DETECTION AND ISOLATION MECHANISMS

3.5.1 *Voter Mechanism*

In this section, the design of the voter module, based on the outcome of the FMEA, is described in detail. To save some hardware resources, the voter module is not connected externally to the NoC routing switch as depicted in Figure 17. Instead, it is embedded into the switch as can be seen in Figure 20 (dashed box). Network packets arriving at an arbitrary input port can be routed to any slot (input buffer) of the voter module. Since the output of the voter module is connected to the switch matrix too, it can be routed to any output port of the routing switch.

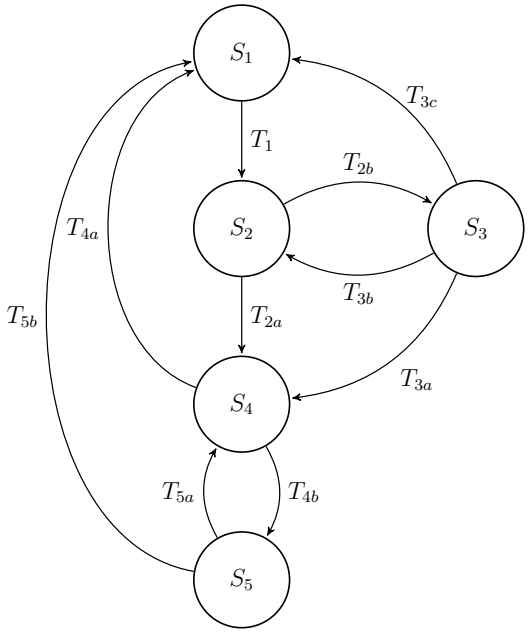

Figure 21: State machine diagram of the voter sub-module

Table 8: Variable Definitions: Voter

Variable	Description
`data_available(3:1)`	not `empty(3:1)`
`min2SlotsOk`	At least two slots are marked as healthy
`min1OkSlotHasData`	At least one healthy slot has data available
`min2OkSlotHaveData`	At least two healthy slots have data available
`allOkSlotsHaveData`	All healthy slots have data available
`allOkSlotsHaveEOP`	An EOP character arrived in all healthy slots
`allOkSlotsVotedOk`	Data in all healthy slots is identical

First, some variables are defined which simplifies the following description of the state machine, see Table 8. Aside from these variables, `voting_ok(3:1)` is a bus provided by the embedded *Word-Voter* [132] which is implemented as shown in Listing 1.

Listing 1: Implementation of Word-Voter

```
voting_ok(3:1) := "111" when dout(1)=dout(2)=dout(3)
                  "011" when dout(1)=dout(2)
                  "101" when dout(1)=dout(3)
                  "110" when dout(1)=dout(2)
din(0) := dout(1) when voting_ok(3:1) = "111" or "011" or "101"
          dout(2) when voting_ok(3:1) = "110"
```

The signal `slot_status(3:1)` is a register which stores the health status of each slot and which can also be updated by an external instance. The registration of the new status is done in several steps. If a write enable signal is active, the new status is temporarily stored and a pending flag is set for each updated slot.

Then, the pending flag register is moved to a temporary register once data is flowing in all pending slots again. Afterwards, a status update flag `reg_status_update` is set for each slot in which an EOP character has been detected. This flag is used by the state machine to finally reactivate the slot in its idle state.

Waiting for the data to flow again is necessary since there might be some delay between the registration of the new status and the arrival of the first valid data. Waiting for the EOP character is necessary due to the integrated *spilling* mechanism: If a slot is marked as faulty, all incoming data in this slot is deleted (or spilled) to prevent other parts of the network from being blocked. After the slot is marked as healthy again, the next arriving packet in this slot will be valid (detectable by an EOP of the current packet). Thus, the spilling must be stopped once a status update has been registered successfully. This is achieved by setting the default value for the `nread` signals as shown in Listing 2.

Listing 2: Default value of `nread` signals

```
for i = 1:3 loop
  nread(i) := slot_status(i) or reg_status_update(i)
end loop
```

STATE S1 This is the idle state in which the state machine remains until a new packet arrives in at least one healthy slot under the premise that at least two slots are healthy. Furthermore, if the slot status register has been updated from an external instance in the meanwhile, the new status is registered now. Pseudo code for the implementation of this state is shown in Listing 3.

Listing 3: Implementation of state S1

```
for i = 1:3 loop
  if register_status_update(i) then
    slot_status(i) := register_status(i)
  end if
end loop
if min2SlotsOk and min1OkSlotHasData then -- Transition T1
  timer := inter-packet timeout value
  state := S2
end if
```

STATE S2 The role of this state is the synchronisation of the incoming data streams. Since data is not taken out of the receive buffers yet, back pressure is applied to the slots where data arrives earlier. While the state machine remains in this state, the inter-packet timeout counter is active. The state is left under two possible conditions: Either data has arrived in all healthy slots or the timeout expired. In the first case, the state machine proceeds to the normal voting operation in state S4. In the latter case, the next action is determined in state S3. Pseudo code for the implementation of this state is shown in Listing 4.

Listing 4: Implementation of state S2

```
timer := timer - 1
if allOKSlotsHaveData then -- Transition T2a
  timer := inter-character timeout value
  state := S4
elsif timer = 0 then -- Transition T2b
  state := S3
end if
```

STATE S3 This state is entered once the timeout has expired in state S2, i.e. not every healthy slot received a packet. The state is left under three possible conditions. If only two slots were healthy before, all slots must be marked as faulty and the state machine moves to idle state S1. If all slots were healthy and two packets arrived, the slot without packet is marked as faulty and the state machine moves on to state S4. If three slots were healthy and only one packet arrived, the slot in which this packet arrived is marked as faulty and the last-resort timeout value is loaded into the timer. Then, the state machine moves back to state S2. Pseudo code for the implementation of this state is shown in Listing 5.

Listing 5: Implementation of state S3

```
if slot_status(3:1) = "111" then
  if min2OKSlotsHaveData then  -- Transition T3a
    for i = 1:3 loop
      slot_status(i) := slot_status(i) and data_available(i)
    end loop
    state := S4
  else  -- Transition T3b
    for i = 1:3 loop
      slot_status(i) := not(slot_status(i) and data_available(i))
    end loop
    timer := last-resort timeout value
    state := S2
  end if
else -- Transition T3c
  slot_status(3:1) := "000"
  state            := S1
end if
```

STATE S4 During the voting or the comparison of the incoming data, the state machine remains in this state. Data is only read from each healthy slot if the transmit buffer of the output port is not full. On the other hand, data is only written to the transmit buffer if at least two healthy slots have data available. As long as some healthy slots have data while some others are empty, the inter-character timeout counter is active. Once all slots have data available, however, the counter is reloaded. This state is left under two conditions: If the current packet was transferred successfully, the state machine moves back to idle state S1. If the Word-Voter flags a mismatch or the time-

out expired, the state machine moves to state S5. Pseudo code for the implementation of this state is shown in Listing 6.

Listing 6: Implementation of state S4

```
for i = 1:3 loop
  if slot_status(i) = '1' and full(3) = '0' then
    nread(i) := '0'
  end if
end loop
if min2OKSlotsHaveData then
  nwrite(3) := '0'
end if
if allOKSlotsHaveData then
  timer := inter-character timeout value
else
  timer := timer - 1
end if
if allOkSlotsHaveEOP then -- Transition T4a
  state := S1
elsif timer = 0 or (allOKSlotsHaveData and not allOkSlotsVotedOk)
     then -- Transition T4b
  last_char_voting_ok(3:1) := voting_ok(3:1)
  state := S5
end if
```

STATE S5 This state is entered if either the inter-character timeout has expired or a voting mismatch has been detected. If only two slots were healthy so far, all slots are marked as faulty and the state machine moves to idle state S1. Otherwise, it is checked if the timeout expired or a voter mismatch occurred. In the first case, the empty slot is marked as faulty. In the latter case, the slot that disagreed with the other slots during voting is marked as faulty. Then, the state machine moves back to state S4. Pseudo code for the implementation of this state is shown in Listing 7.

Listing 7: Implementation of state S5

```
if slot_status(3:1) = "111" then -- Transition T5a
  if timeout expired then
    for i = 1:3 loop
      slot_status(i) := slot_status(i) and data_available(i)
    end loop
```

```
  else
    for i = 1:3 loop
      slot_status(i) := slot_status(i) and last_char_voting_ok(i)
    end loop
  end if
  state := S4
else -- Transition T5b
  slot_status(3:1) = "000"
  state := S1
end if
```

3.5.2 *Multicast Mechanism*

Multicasting data to redundant stream processors is done in a distributed manner within the routing switches. As mentioned in Section 3.4, the multicast mechanism must be of non-blocking nature to handle faulty processors that block incoming network traffic. A conceptual circuit diagram of the non-blocking multicast mechanism is shown in Figure 22. Say, a network packet arrives at port 0 and its logical address is assigned to physical port 1, 2 and 3. Then, the multicast mechanism will transfer a data character to port 1, 2 and 3 if the receive buffer of port 0 is not empty and all transmit buffers of port 1, 2 and 3 are not full. This is done by using the handshake signals `full`, `empty`, `nread` and `nwrite` as can be seen in Listing 8.

Listing 8: Implementation of multicast handshake signals

```
minOneFull  := (full(1) and multicastEn(1)) or
               (full(2) and multicastEn(2)) or
               (full(3) and multicastEn(3))
nwrite(1:3) := empty(0) or minOneFull
nread(0)    := minOneFull
```

To tolerate potentially faulty stream processors, a multicast timeout mechanism is used which is always active if one and only one output port of all active output ports is full, see Listing 9.

Listing 9: Implementation of multicast timeout mechanism

```
timerReload := true if not
               only_one_set(full(1:3) and multicastEn(1:3))
```

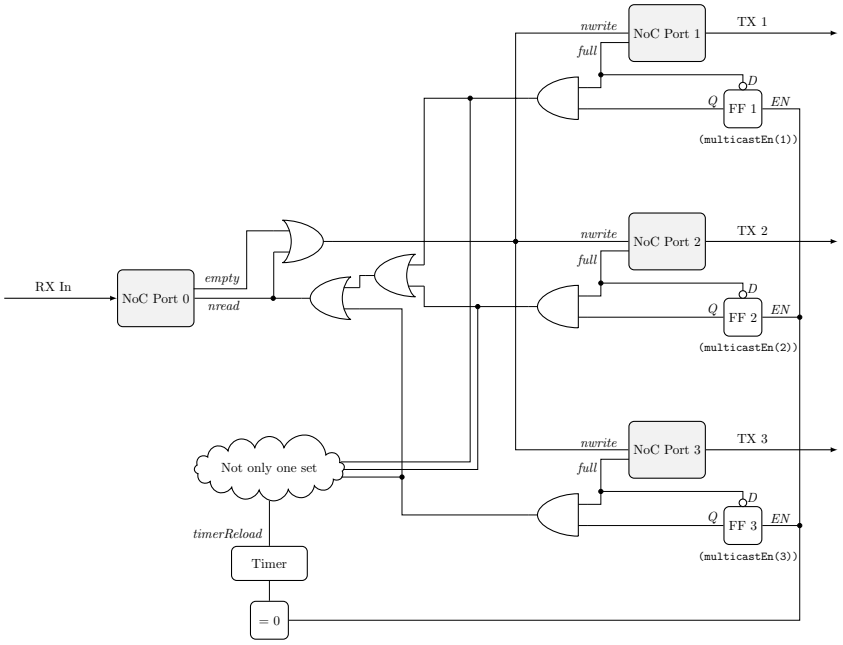

Figure 22: Circuit diagram of the multicast mechanism

If this timeout elapses, it is assumed that the stream processor, which is associated with the blocking output port, is faulty. Then, the output port is removed from the current multicast round by setting its `multicastEn` flag to zero. As a result, the remaining redundant stream processors receive input data, although some additional latency, equal to the multicast timeout period, must be expected.

3.5.3 *Addressing Scheme*

The proposed NoC routing switch comprises a routing table, which can be programmed by sending configuration network packets to an internal configuration port. The same configuration port is also used to program the timeout values described in the previous sections.

The routing switch can handle two types of addresses: (i) physical addresses (range 1 to 31) that correspond to the number of the routing switch ports and (ii) logical addresses (range 32 to 255) that can freely be mapped to routing switch ports. Reserved address 0 is used to reach the internal configuration port.

Within the network topology, each partition has a hard-coded logical address, i.e. even if a partition can host different types of stream processors, the logical address always remains the same for this partition. Similarly, the embedded voter module and the internal config-

uration port have their own fixed logical address. There are several good reasons why the address is mapped to the partition rather than the stream processor, e.g.:

- The FDIR framework remains application-independent. Only the predefined partitions in the network require a logical address. Which stream processors are placed on these partitions later on does not matter.
- It is not required to re-synthesise the Hardware Description Language (HDL) code of redundant stream processors for different addresses.

By programming the routing table, the logical addresses can be mapped to output port(s) of the routing switch. If a logical address is mapped to more than one output port, the multicast mechanism becomes active as described in Section 3.5.2. As a consequence, any desired data flow within the network, including multicasts, can easily be implemented by simply reprogramming the routing tables within the routing switches.

Aside from the already mentioned failure modes, the FMEA also revealed that in some cases the logical or physical address of a network packet could be falsified due to a failure in the stream processor. This failure mode is a serious hazard for a network as it can lead to congestion if a packet is routed to the wrong part of the network. This is especially true for babbling idiots, which transmit indefinitely long network packets.

Regarding failure isolation, two cases must be distinguished. If the logical or physical address of a faulty network packet is not falsified, it will find its way to the next voter module. Within the voter module, the faulty packet is masked out and therefore the failure is successfully isolated. However, if the address is falsified, the packet must be removed from the network as soon as possible to avoid any congestion.

To achieve this, the following simple but powerful failure isolation mechanism is added to the routing switch:

- Aside from mapping a logical address to one or more output ports, it can also be restricted to one single input port. Then, any faulty packet taking on the identity of another packet will be deleted by the routing switch if it does not arrive at the input port, to which the logical address was restricted to. In addition,

any packet with a logical address that is not stored in the routing table is deleted by the switch. Ultimately, the combination of these two mechanisms ensures that packets with wrong addresses are deleted by the routing switch, in which they try to take a route they are not allowed to.

- Similarly, physical addresses can be falsified. In normal operation, physical addresses are not needed for the data flow because all communication within the FPGAs is based on logical addressing. They are, however, required during the initial configuration of the system because the routing tables cannot be programmed without physical addresses. To achieve failure isolation, another mechanism is added that allows physical addressing to be switched off for each port of the routing switch. Therefore, physical addressing can simply be disabled for all routing switch ports after initial configuration.

3.6 DATA RESYNCHRONISATION

An often-mentioned problem in connection with modular redundancy is the required data resynchronisation between the redundant module instances after a module has been successfully repaired. Fortunately, typical payload processing applications do not depend on too many state variables. The number of state variables required for the initialisation of a processor after reset is often limited to a handful of configuration and feedback variables. For instance, an image compression core might need a variable storing the compression quality and one storing the image line width. Another example could be an encryption algorithm used in some feedback mode. Here, a feedback variable storing the last cipher text might be needed to initialise the freshly repaired processor.

Assuming that the processing chain has more network bandwidth available than the input data stream or alternatively, that well dimensioned buffers are available in the network, the data processing could be stopped for a short time period in which the state variables are shared between the currently functional stream processors and the freshly repaired stream processor. It is proposed to use the already available resources in the FDIR methodology presented here to accomplish this task.

As can be seen in Figure 16, each stream processor already comprises a state variable memory, which stores all required initialisation

variables externally to the embedded IP core. Thus, it is sufficient to dump theses variables over the network to the freshly repaired processor, which can then store this variable set in its own state variable memory. To increase the reliability of this mechanism, the state variables of the two functional processors could first be compared before the freshly repaired processor registers them. Since each voter module also works as comparator, an elegant solution would utilise the voter module for this task. Consider the example shown in Figure 17. Say, the processor connected to routing switch 1 has been just repaired and needs to be updated with initialisation variables. The other two processors could stop the data processing after finishing the processing of the current block of data and send their state variables to voter module **V**. The voter module could compare the network packet, which contains the state variables, and forward all identical state variables to the freshly repaired processor which then updates its own memory. However, there are two main issues that must be taken into account:

1. The two functional stream processors are not running synchronously and therefore a synchronous request to dump the state variable memory could lead to situations where one processor is dumping newer and hence other variables than the second processor.

2. The two functional stream processors must stop any data processing until the freshly repaired instance has updated its own state variable memory and resumed its operation. Otherwise, the shared variables might be already invalid once they become active in the freshly repaired processor.

To solve the first problem, the request to dump the state variables must be injected into the input data stream. For instance, a small hardware module placed at the beginning of the processing chain could send out a small synchronisation request packet. This request packet would traverse the network like the regular network stream. Relative to this input data stream it would arrive at the same bit position and thus it could be ensured that the still functional processors receive this request packet when they are both in the exact same state. Then, the functional processors could bundle their state variables into a synchronisation packet, which is attached to the output data stream.

At some point, the voter module would receive the redundant synchronisation packets. The voter module is able to detect this special

kind of packet and would move into a resynchronisation mode. While in this mode, the aforementioned second issue could simply be solved by applying backpressure to the slots associated with the two functional processors. In other words, after receiving the synchronisation packets, no more data characters are taken out from the slot buffers and therefore the functional processors would be forced to stop the data processing (with some latency as the buffers in the network path would fill up first). In addition, the voter module could start a data-synchronisation timeout. The timeout period must be chosen wisely to (i) give the freshly repaired module enough time to update its state variable memory but (ii) also take the buffer sizes and bandwidths within the network into account. The voter module would then send the synchronisation packet to the freshly repaired stream processor. If no comparison mismatch occurred, the voter module would go into a special wait state afterwards. A short time later, the synchronisation packet would arrive at the freshly repaired processor, which would update its state variable memory and resume operation. Once the first data is processed, its first output packet would arrive at the voter module, which is still in its wait state applying backpressure to the other two redundant processors.

Now, the voter module would reintegrate the freshly repaired stream processor because the first output packet of the freshly repaired processor would be identical to the output packets of the other two redundant processors. It would stop the backpressure and resume normal operation. However, if the data-synchronisation timeout elapsed, it would be assumed that something went wrong during resynchronisation. In this case, the backpressure would be released and normal operation would be resumed without reintegrating the freshly repaired processor.

3.7 CONCLUSIONS

The novel Distributed Failure Detection technique presented in this chapter offers many advantages compared to classic mitigation approaches for SRAM-based FPGAs. On the one hand, the methodology is adaptive, i.e. redundancy can be added or removed during flight to either increase the system availability or decrease the power consumption of the system. On the other hand, by utilising a NoC as communication architecture, redundant stream processors can be distributed over several FPGAs and hence, multi-FPGA systems can be utilised

more efficiently. Although data resynchronisation after repair is a serious issue, a novel resynchronisation scheme was proposed, which results in low implementation complexity and area overhead. Furthermore, the design of an innovative voter module has been presented. In fact, the module is much more than just a majority voter: Firstly, it integrates into a network architecture which allows the voting of processor instances that can freely be distributed over the network. Secondly, it is able to synchronise incoming network streams and can cope with all failure modes that typically emerge in such streams. Finally, it can signal the status of each slot to an external supervisor and can automatically reintegrate repaired processors. In addition, an appropriate multicast mechanism and addressing scheme is proposed, which guarantees a fault-free routing of data streams through the network. All the aforementioned mechanisms are integrated in a custom-designed NoC routing switch.

The FDIR methodology is not technology-dependent and could be applied in a similar way to other systems consisting of multiple processing elements interconnected in a modern switched fabric network.

4

CHAPTER 4: AVAILABILITY ANALYSIS METHOD

4.1 INTRODUCTION

SRAM-based FPGAs suffer from radiation induced failures, see Section 2.2. However, the device can be recovered from these failures by one of the configuration memory refresh methods reviewed in Section 2.4.1. Therefore, a combination of a failure detection and a recovery technique makes this type of FPGA a repairable system. The probability that a repairable system functions correctly is called *steady state availability*, often defined as:

$$A = \frac{T_m}{T_m + T_d} \tag{1}$$

where T_m is the uptime and T_d the observed downtime, which is the sum of the time required to detect and recover failures.

Availability analysis is an integral part of space product assurance procedures and a dedicated European Cooperation for Space Standardization (ECSS) standard is available for European space projects [133]. This standard gives several definitions for availability. In the following the term availability stands for *inherent* steady state availability, i.e. downtime during idle periods does not contribute to the overall system availability. According to the ECSS standard, important objectives of availability analysis include the verification of whether or not a system conforms to the availability requirements. In addition, it also identifies unavailability contributing factors in order to quantify their impact on (i) the decision-making process, and (ii) the risk evaluation, reduction and control. The implementation of such an availability analysis method with regards to SRAM-based FPGAs has two significant advantages:

- It gives an estimation of the number of failure occurrences that are to be expected either during a specific mission time frame or the overall mission lifetime.
- It allows the comparison of different redundancy and recovery schemes, which is especially important for applications where

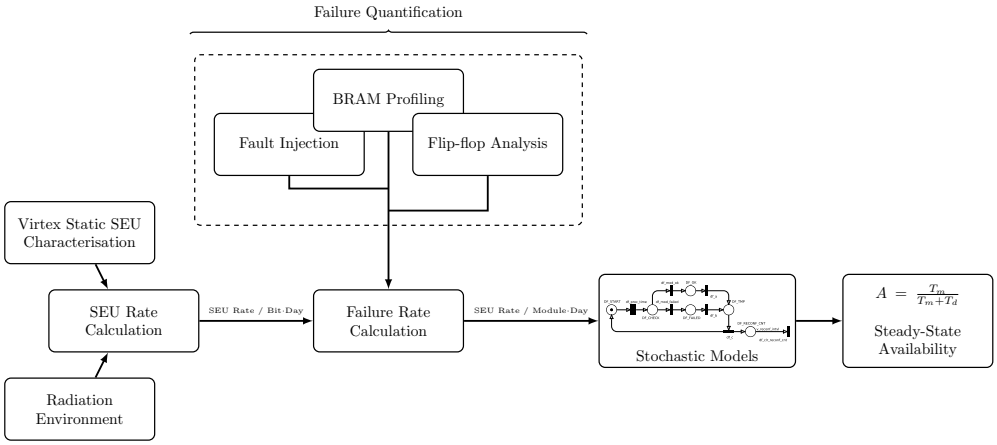

Figure 23: Block diagram of the proposed availability analysis methodology

an optimal mitigation approach must be found trading-off power, area and reliability overheads.

In this chapter, a methodology is presented that allows the availability analysis of stream processors, implemented on SRAM-based FPGAs. In Section 4.2, an overview of the proposed methodology is given. Section 4.3 describes how the number of sensitive memory elements can be quantified, including configuration memory, Block RAM and flip-flop cells. In Section 4.4, stochastic Petri net models for the most important FDIR configurations are proposed. Finally, Section 4.5 concludes the chapter by outlining the proposed novelties.

4.2 OVERVIEW

A block diagram of the proposed methodology is depicted in Figure 23. First, the SEU rates per bit-day are determined for the configuration memory, the Block RAMs (hereafter also referred to as RAMB16) and the flip-flops of a particular SRAM-based FPGA device in a specific orbit. In the following, a Virtex-4 FPGA device is used as an example. The calculations are based on static SEU characterisation data that was gathered from accelerated radiation testing and published by Xilinx and NASA [3]. The radiation environment, i.e. the heavy ion and proton fluxes of the orbit, is calculated using radiation models standardised in the European standard ECSS-E-ST-10-04C [134]. The calculated SEU rates give the probability of a bit flip in one single memory element.

However, to compute the availability of a stream processor, the SEU rate per day and stream processor must be known and not just the rates for the memory elements. Thus, the bit upset rates must be scaled by the number of sensitive memory elements that must first be determined. The sensitivity of the configuration memory is quantified by randomly injecting faults into the memory using a fault injection system whereas the sensitivity of the Block RAMs is estimated using a custom-built memory profiling tool. For the usually small number of flip-flops it is simply assumed that all of them are sensitive. Once the sensitivity of all memory types is quantified, the SEU rate for the whole stream processor is known.

To analyse the availability, the chosen failure recovery approach must also be considered. The typical recovery approach for the configuration memory is often referred to as *scrubbing*, a mitigation technique that refreshes the memory content from time to time with a correct bitstream, see Section 2.4.1. In contrast to partial reconfiguration, scrubbing does not affect the user memory and can thus be executed during circuit operation. For some failures, however, such a memory refresh is not sufficient because the failure could have already manifested itself in the user logic (e.g. state machines, counters and similar state-dependent logic). Then, the user logic must additionally be reset. For user memory (RAMB16s and flip-flops), two basic failure modes exist: (i) a failure can either propagate to an output of the FPGA or (ii) it can get trapped in a feedback loop. In the first case, the failure is of a transient nature only and no further recovery actions are required. In the latter case, the user logic must be reset to a safe initial state.

If one wants to model the availability of a processor to which a specific recovery approach is applied, e.g. frame-based scrubbing or partial reconfiguration, the SEU rates for the aforementioned failure modes must be calculated separately. For instance, some configuration memory bit upsets only require a memory scrub, while others need an additional circuit reset. Therefore, the SEU rates for all these cases must be determined. This is accomplished for the configuration memory by the fault injection system that automatically tries to recover from a failure in several ways and is thus able to classify and quantify the sensitive bits, described in Section 4.3.1. For the RAMB16 cells and flip-flops, a custom-built netlist parser tool analyses how many of these memory elements propagate into feedback loops and how many of them do not. Then, the SEU rates for both cases can be calculated.

Once all SEU rates of interest are available, the steady-state availability is calculated using *Stochastic Petri Nets*. Several models are proposed, which take different redundancy and failure recovery approaches into account but in principle, this modelling technique is flexible enough to support all kind of FDIR approaches.

4.3 QUANTIFICATION OF SENSITIVE MEMORY ELEMENTS

The following three main types of memories, available in radiation-tolerant SRAM-based FPGAs, must be considered in a detailed availability analysis:

- Configuration memory: The configuration of the FPGA is stored in volatile SRAM memory cells. A part of the memory stores control bits, which define the routing resources and the content of the LUTs. Thus, a radiation-induced upset in one of these sensitive memory elements can manipulate the circuit in such a way that a failure becomes measurable at its output.
- Block RAM: In many streaming applications, a large amount of embedded RAM blocks is utilised by the user logic. If a value, which was falsified by an upset, is read from such a memory, it can manifest itself as a failure and either propagate to the output of the circuit or get trapped in a feedback loop within the circuit.
- User flip-flops: In most cases, flip-flops do not contribute as much as the aforementioned memory types to the overall cross-section due to their typically low number. However, since flip-flop upsets can get trapped in feedback loops too (which requires some failure recovery action), they are not neglected for the proposed availability analysis method.

4.3.1 *Configuration Memory (via Fault Injection)*

Although analytic tools for the quantification of sensitive configuration memory bits are available, see Section 2.5, fault injection is probably still the most reliable technique for this task due to its capability to take into account the dynamic behaviour of the application. With this capability, the following steps can be accomplished:

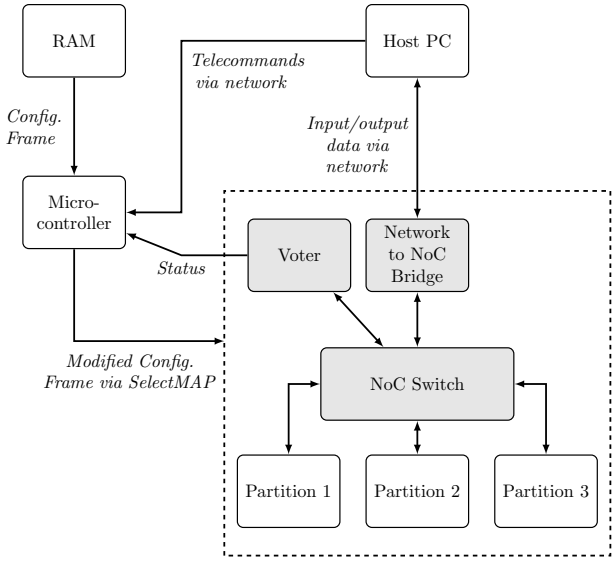

Figure 24: Fault injection system with an SRAM-based FPGA device under test (dashed box)

- The quantification of three types of sensitive configuration bits. Those which lead to failures that:
  - can be scrubbed ($F_{C/S}$).
  - need an additional circuit reset after scrubbing ($F_{C/SR}$).
  - can only be repaired by a repeated partial reconfiguration of the stream processor ($F_{C/Re}$).
- The creation of a database that stores information about configuration bits, for which the failure mode is exactly known. The database can later be used to validate FDIR techniques.

As part of this PhD work, a fault injection system is proposed as outlined in Figure 24. The FPGA is divided into three partitions hosting three identical stream processors, all interconnected using a NoC routing switch. The proposed voter module is part of this NoC routing switch, see also Section 3.5. A network bridge allows the communication between external components and each partition via a network. The fault injection campaign is controlled by two instances:

- An embedded software running on a microprocessor is responsible for the fault injection itself. This is done by first retrieving a desired single configuration frame from RAM, to which the partial bitstream under investigation has been copied before. Then, a bit in this configuration frame is flipped before it is finally

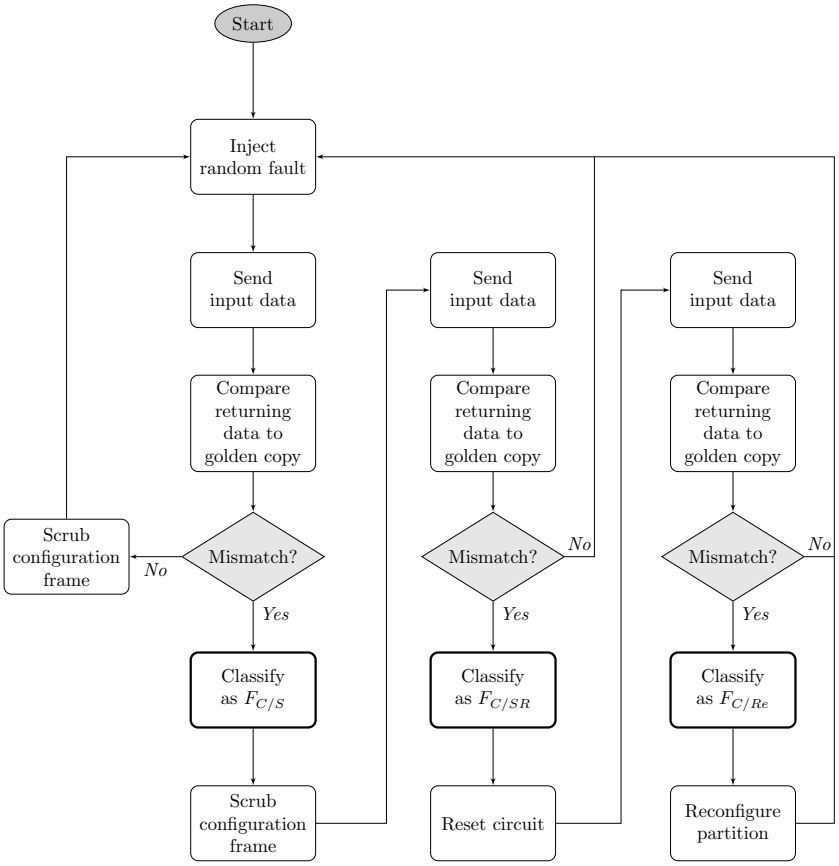

Figure 25: Flow chart of the novel fault injection and failure classification algorithm

downloaded to the FPGA via one of its configuration interfaces, e.g. SelectMAP.

- A software running on a host PC controls the fault injection campaign by sending telecommands to the embedded software system.

A flow chart of the novel fault injection and failure classification algorithm can be seen in Figure 25. After injecting a fault into the partial bitstream of one of the stream processors, the host PC software sends a full input data block to the FPGA. The data block is multicast to the three identical stream processors within the NoC routing switch. The returning data from the stream processors is routed through the voter module, which is used as a failure detector, and then returned to the host PC software. The host PC software requests the status of the voter module from the microcontroller to determine if the injected fault led to a failure. If this is not the case, the configuration frame is scrubbed to avoid an accumulation of faults in the configuration memory and the next random fault is injected. If there was a mismatch, however, the sensitive bit is classified as $F_{C/S}$. The configuration frame is scrubbed and the host PC software sends another data block to the stream processors to check if the scrubbing was successful. If so, no additional actions are required and the next random fault can be injected. If not, the classification of the sensitive bit is updated to $F_{C/SR}$ and the circuit is reset by the embedded system (again by sending an appropriate telecommand). Then, the procedure is repeated. If the reset was successful, the next random fault is injected. Otherwise, if the stream processor is still not returning a correct answer, the classification is updated to $F_{C/Re}$. The classification for each injected bit is stored in a database for later analysis.

Faults are injected just before a data block is sent to the stream processor. In reality, however, a fault could also occur during processing. Preceding fault injection was chosen intentionally as it represents a worst-case scenario that maximises the detection coverage of sensitive bits.

The automatic fault injection technique can be rather slow since full data blocks (e.g. full images) are sent through the stream processors. However, this is important to make sure that all of the states of the circuit are traversed. It might take some time until the fault manifests itself as a failure by propagating to the circuit's output and as a consequence, the failure could stay undetected if the state space

is not fully covered. The technique allows a very detailed classification of the different failure modes, using a real application. It is also non-intrusive, i.e. no circuit modifications are required for the fault injection campaign. Therefore, a fully placed and routed design could be analysed in a very late design phase, an important aspect for the typical qualification process in space engineering projects. And even if only a limited number of faults can be injected, useful results can still be obtained by using statistical theory. Since the outcome of the fault injection campaign has a binomial distribution (recall the classic urn problem), the number of sensitive configuration bits can be estimated by a rather low number of statistical samples. To get an idea how many samples are necessary, the formula for Wald confidence intervals [135] can be used:

$$p = \hat{p} \pm z_{1-\frac{1}{2}\alpha} \cdot \sqrt{\frac{1}{n}\hat{p}(1-\hat{p})} \tag{2}$$

where $\hat{p}$ is the proportion of success (e.g. the ratio of sensitive bits to all bits), n the number of trials, and z the $1-\frac{1}{2}\alpha$ quantile of a standard normal distribution with α as error percentile.

Once the fault injection results are available, a more conservative approach for the final estimation of the proportion is chosen. The so-called Clopper-Pearson interval is known to have never less than the given coverage rate [135] which makes it a good candidate for worst-case estimations.

4.3.2 *Embedded Block RAM (via Memory Profiling)*

Streaming applications often utilise huge amounts of embedded Block RAM elements, e.g. as First In, First Out (FIFO) buffers or ROM. As a consequence, the number of sensitive bits inside these elements can be rather large, sometimes comparable to the total number of sensitive configuration bits. Therefore, upsets in Block RAMs cannot be neglected in an availability analysis. However, the estimation of the Block RAM utilisation is complicated if not impossible with the standard toolchain. If only the absolute number of Block RAMs is taken into account, the number of susceptible bits would be way too overestimated. In reality, the number of sensitive bits depends on the dynamic behaviour of the application. Firstly, many rows within a Block RAM are simply not used. Secondly, faults affecting memory rows

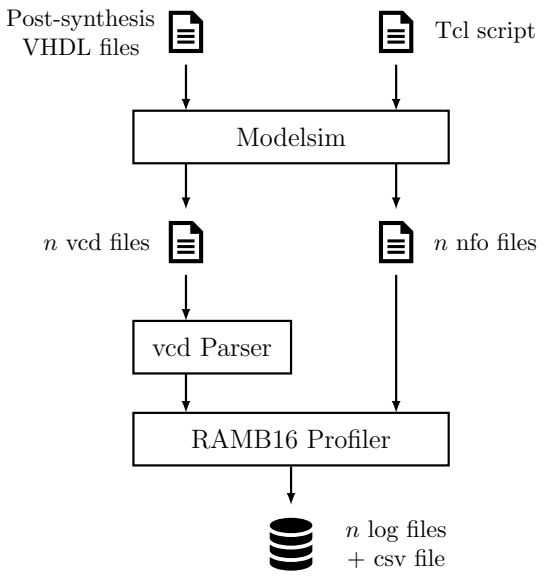

Figure 26: Block RAM profiling tool

which are not read out after the fault occurred will also not lead to a failure.

In the course of this work, a novel tool for the dynamic profiling of Block RAMs has been developed. The basic idea, as outlined in Figure 26, is as follows: The processing of one full data block is simulated using a post-synthesis simulation model, e.g. in ModelSim [136]. During simulation, a `vcd`-file is created for each Block RAM which records the changes on the address and write enable lines of both ports for the whole simulation run. Furthermore, some information about the Block RAM is stored in a `nfo`-file, including the read and write width configuration, the write mode (write-first, read-first, no-change) as well as a flag which signals if the data in and data out lines of both ports are actually connected. All this information is gathered and stored automatically by a Tcl script, which is executed within the simulator.

Once all files are available, another tool parses the `vcd`-files into C++ objects where the time-value pairs of the signals are stored. Then, the memory profiling begins: The tool steps through the simulation run to identify read and write accesses. This is accomplished by searching for signal changes in the address and write enable lines of connected ports. While write accesses can easily be identified by the corresponding write enable signal, read accesses must be guessed. This is done by interpreting an address change, which occurs while the write enable signal is low, as a read access. The approach is slightly conservative but without any further knowledge about the circuit, it must

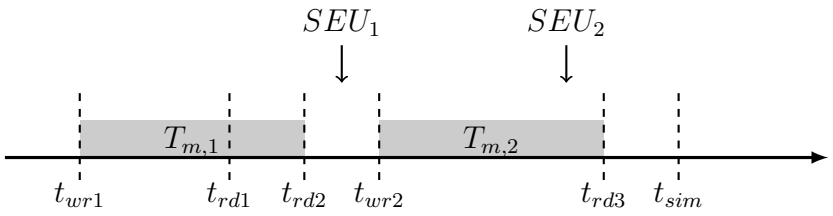

Figure 27: Example for the calculation of the correction factor

be assumed that any valid data at the output of a Block RAM is also used in the design.

The memory profiling tool has three main tasks. Firstly, it must identify the real utilisation of the Block RAM. Since the tool keeps track of the memory addresses that were accessed during the simulation run, the number of used memory bits can easily be calculated. Secondly, it must identify which memory rows, respectively addresses, are used as ROMs and which are used as RAMs. This is achieved by interpreting rows, which are only accessed for reading but not for writing, as ROM rows. Thirdly, the tool must determine a correction factor for each RAM row that takes the time spans into account in which a fault cannot manifest itself as a failure.

Take the timeline in Figure 27 as an example, where some read and write accesses of one memory address are shown. The first fault SEU_1 occurs between a read access t_{rd2} and a subsequent write access t_{wr2}. Since the memory row is overwritten with a new value, the fault cannot manifest itself as a failure. The second fault SEU_2, however, manifests itself as a failure because the memory row is read out at t_{rd3}. All N time spans $T_{m,n}$ in which a memory row m is susceptible (grey boxes in Figure 27) are first accumulated by the memory profiling tool. Then, the results of all M memory rows are averaged. Finally, dividing the averaged value by the total simulation time leads to the correction factor τ_S:

$$\tau_S = \frac{\sum^{M} \sum^{N} T_{m,n}}{M \cdot t_{sim}} \tag{3}$$

τ_S can only be calculated for RAM rows because obviously, ROM rows are never overwritten by the user logic. RAM rows in Block RAMs that are configured in read-first mode must be handled in a similar way: In this mode, a memory row is read out just before the write access and thus the whole time span between the last and the

current write access must be assumed to be susceptible to upsets, i.e. $\tau_S = 1$.

The memory profiling tool outputs one `log`-file per Block RAM with detailed profiling information and one `csv`-file with a summary of all results.

With the profiling tool, the number of susceptible Block RAM bits can be quantified. However, the quantification of the failure modes is not done yet. As mentioned earlier, a failure can either propagate to an output of the FPGA or it can get trapped in a feedback loop. While no special recovery action is necessary in the first case, the latter failure mode necessitates a circuit reset. To get an idea how many of the failures propagate into feedback loops, a netlist parser was developed. The tool takes an `edif`-file as input and parses the netlist into a directed graph of C++ objects. Then, the tool is able to traverse the graph using a Depth-First Search (DFS) algorithm. The netlist is flattened in such a way that the graph only comprises primitives (flip-flops, RAMB16s etc.) as vertices. Within each vertex, all inputs are connected to all outputs. In other words, it is assumed that a failure, which arrives at an input of a primitive can propagate to any output of that primitive.

The implemented algorithm takes a set of primitives, here RAMB16 blocks, as starting points. From the starting points, all possible edges and vertices are explored until either (i) an output pin of the FPGA is found or (ii) an input pin of a primitive is found which was already visited in the current exploration. In this case it is clear that the failure got caught in a feedback loop. If at least one output pin of the source primitive leads to a feedback loop, the second failure mode (necessitating a reset) is assumed for that primitive.

4.3.3 *User Flip-Flops*

The number of user flip-flops can easily be determined using the standard toolchain or alternatively, by counting the flip-flop primitives in the netlist graph. But again, it is unclear how many of the flip-flops propagate into feedback loops. Therefore, the netlist parser tool is used a second time, using flip-flops primitives as starting points rather than Block RAM primitives. As for the Block RAMs, if at least one output pin of the source primitive leads to a feedback loop, the second failure mode (necessitating a reset) is assumed for that primitive.

Table 9: Possible recovery actions for all failure modes

Failure Mode	Recovery Action
$F_{C/S}$	Reconfiguration OR Scrubbing
$F_{C/SR}$	Reconfiguration OR (Scrubbing AND Reset)
$F_{C/Re}$	Reconfiguration
F_{ROM} (ROM cells)	Reconfiguration
F_{RAM} (RAM cells)	Reconfiguration OR Reset
F_{FF} (flip-flops)	Reconfiguration OR Reset

4.4 AVAILABILITY ANALYSIS

4.4.1 *Radiation Environment*

First, the bit upset rates for the configuration memory, the Block RAMs, and the flip-flops in a particular radiation environment must be determined. The required heavy-ion and proton fluxes are calculated according to ECSS-E-ST-10-04C [134]: For cosmic rays, the ISO-15390 GCR model is used while solar minimum conditions are assumed. For trapped electrons, the AE8MAX model is used for low Earth orbits (LEO) and the MEOv2 model for medium Earth orbits (MEO). For trapped protons, the AP8MAX model is used for both, LEO and MEO. The fluxes during Solar Particle Events (SPEs) are calculated using the CREME96 Worst Case 1 Day solar flare model. All fluxes are averaged over 100 orbits and thus take any anomalies into account. Common practice is followed by assuming 100 mil of solid spherical aluminium shielding, although it was shown in the past that this assumption is an underestimation for most spacecraft electronics boxes [137].

4.4.2 *Stochastic Petri Net Models*

For availability modelling, stochastic Petri nets are used that can analytically be solved with the TimeNET 4.1 tool [138]. In the following, several nets are proposed that can estimate the availability of a stream processor in a particular FDIR configuration. Three basic configurations are covered, see Figure 28. In configuration (a), a stream processor is used without redundancy. Since no failure detector exists in this configuration, failure recovery must be done in a preventative manner. If the data is processed in bursts, it is proposed to do periodic

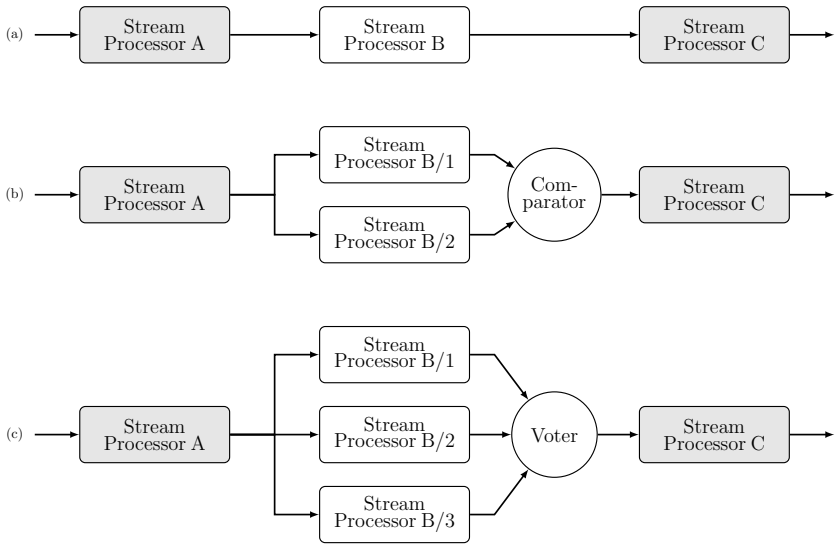

Figure 28: Example of a typical satellite payload streaming architecture with a stream processor in different redundancy configurations: (a) No redundancy, (b) Duplication with Comparison, (c) Triple Modular Redundancy

partial reconfiguration between the bursts (approach 1). To further increase the availability, this approach can be combined with additional periodic scrubbing (approach 2). In configuration (b), a stream processor is duplicated and a comparator is used as failure detector. Once a failure is detected, both stream processors are reconfigured (approach 3). In configuration (c), a stream processor is triplicated and a majority voter is used for failure detection and masking. Once a failure is detected, the faulty stream processor is reconfigured (approach 4).

First, however, the total number of estimated sensitive bits of each failure mode must be multiplied by the corresponding bit upset rate (for F_{FF}, it is assumed that half of the flip-flops is storing a logical 1 and the other half a logical 0). Then, since the stochastic Petri net method requires Mean Time Between Failure (MTBF) values rather than failure rates, the inverses of the resulting failure mode upset rates are used as input parameters for the models.

The MTBF value for a specific failure mode F_x is calculated as follows:

$$\mathrm{MTBF}(F_x) = (n_x \cdot B_x)^{-1} \tag{4}$$

where n_x is the number of sensitive memory elements for the specific failure mode and B_x the upset rate per bit-day for this specific type of memory element in the targeted radiation environment.

EXAMPLE 500,000 sensitive configuration memory elements for failure mode $F_{C/S}$ (failures that can be repaired by scrubbing) were found for a specific stream processor using the aforementioned fault injection method. The FPGA will fly in an orbit for which a configuration memory cell bit upset rate of $7 \cdot 10^{-7} (\text{bit} \cdot \text{day})^{-1}$ was determined. Then, the MTBF value for this failure mode is:

$$\text{MTBF}(F_{C/S}) = (\frac{7 \cdot 10^{-7}}{\text{bit} \cdot \text{day}} \cdot 500,000 \text{ bits})^{-1} = 2.9 \text{ days} \tag{5}$$

4.4.2.1 *No Redundancy*

First, the availability of a single stream processor is investigated. Without redundancy, no reliable failure detector mechanism exists and as a consequence, failure recovery must be done in a preventative manner, i.e. without any knowledge about the health status of the processor. A strategy solely based on scrubbing cannot be recommended due to the failures which can get trapped in the user logic. Consulting Table 9, two approaches can be applied, depending on how the input data is streamed into the processor:

APPROACH 1 - PERIODIC PARTIAL RECONFIGURATION If the input data is delivered in bursts, e.g. 500 image frames, and there is some time between the bursts to do a partial reconfiguration, it is recommended to do such a periodic reconfiguration just before the processing of each burst. The benefit of this solution is its simplicity and the fact that the processor can be recovered from all possible failure modes in one go.

The stochastic Petri net model has three basic nets that are linked to each other via logical conditions. For the aforementioned case, the first net is shown in Figure 29. The net models the health status of the processor: If the token is on place `MOD_OK`, the processor is working correctly. If one of the failure modes occurs (modelled by the exponential transitions `mod_seu_*`), the token moves into state `MOD_F1`. The failure recovery process is modelled by the immediate transition `mod_reconfig` which is only enabled when a trigger condition in the second net becomes true.

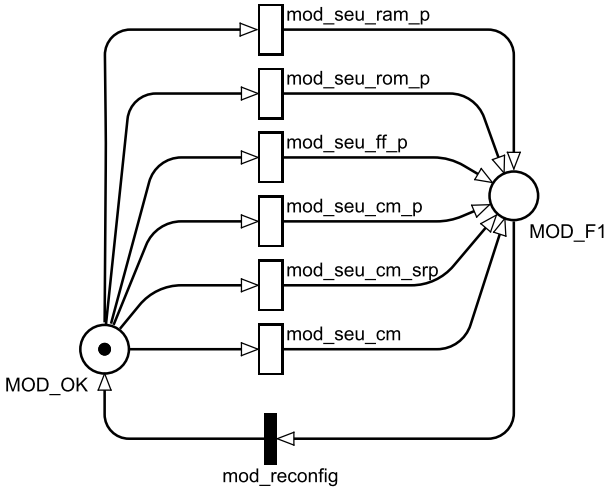

Figure 29: Stochastic Petri net no. 1 (approach 1)

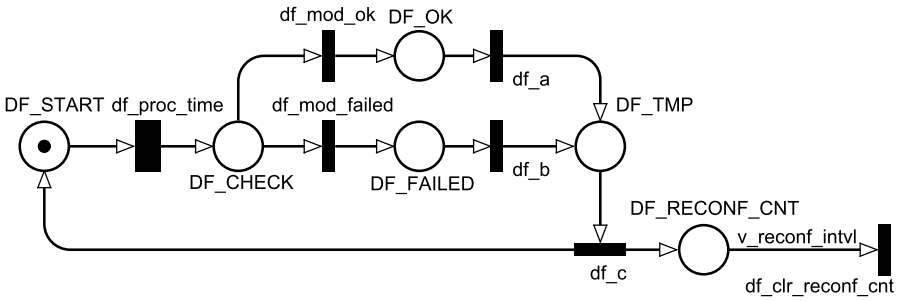

Figure 30: Stochastic Petri net no. 2 (approach 1)

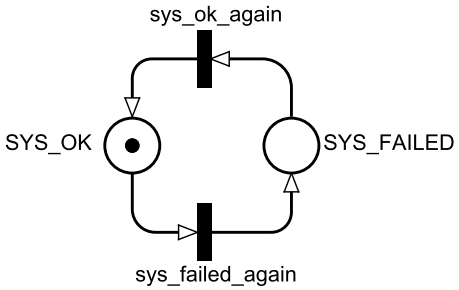

Figure 31: Stochastic Petri net no. 3 (approach 1)

The second net, as depicted in Figure 30, models the data flow through the processor. A token represents the data block. Transition `df_proc_time` is deterministic and models the processing time of the data block. After processing, the first net is checked for the health status and the token moves either to place `DF_OK` or `DF_FAILED`. Then, the token moves back to the start place `DF_START` and another stochastic experiment begins. On the way back to the start place, the token is duplicated and one copy of the token moves to `DF_RECONF_CNT`. This place is a counter which counts the number of processed data blocks. If the reconfiguration interval threshold value is reached, the counter place is cleared and the failure recovery trigger condition becomes true.

A third net is used for the modelling of the overall processor availability. The currently processed data block can either be counted as OK or failed, see Figure 31. The transitions in this net are only enabled when the token is on place `DF_OK` or `DF_FAILED` in the second net. Ultimately, the steady state availability of the processor is determined by calculating the probability that the token is on place `SYS_OK`.

APPROACH 2 - PERIODIC RECONFIGURATION AND SCRUBBING

If the burst length is long or the probability of $F_{C/S}$ is rather high, it might be worth to combine the periodic partial reconfiguration with a more frequent periodic scrubbing. The Petri nets must slightly be modified to model this case. In the first net, see Figure 32, tokens moving via `mod_seu_cm` arrive at a new place called `MOD_F2`. From there the token can leave back to `MOD_OK` via a new deterministic transition for scrubbing called `mod_scrub`, analogous to `mod_reconfig`. To trigger this transition, an additional counter called `DF_SCRUB_CNT` is added to the second net, see Figure 33, parallel to `DF_RECONF_CNT`.

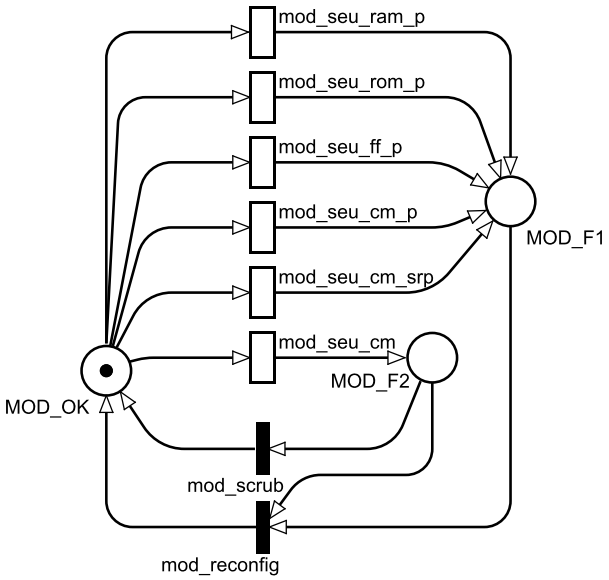

Figure 32: Stochastic Petri net no. 1 (approach 2)

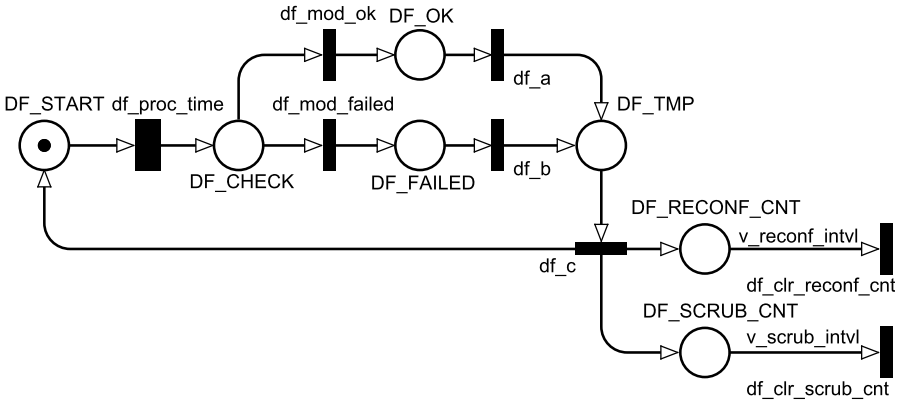

Figure 33: Stochastic Petri net no. 2 (approach 2)

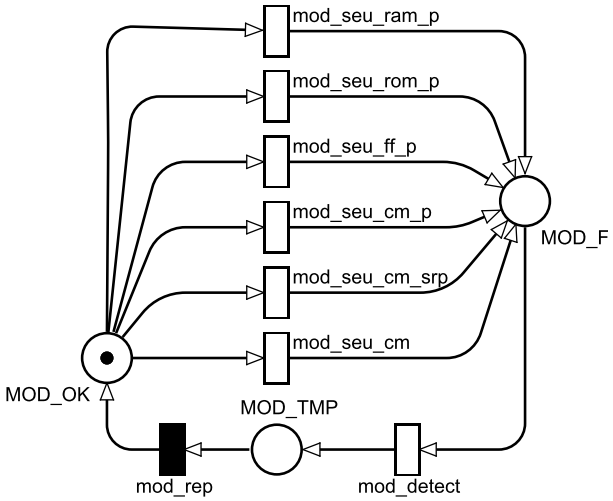

Figure 34: Stochastic Petri net no. 1 (approach 3)

4.4.2.2 *Redundancy with On-Demand Reconfiguration*

If the stream processor is duplicated or triplicated, a comparator or voter module, respectively, can be used as failure detector. Then, the failure recovery process can be triggered on demand. Since the processor is immediately reconfigured and does not have to wait for the next recovery cycle, the availability increases.

APPROACH 3 - DUPLICATION WITH COMPARISON With two redundant stream processors, the Petri net model can be simplified because the repair process becomes independent of the data flow through the stream processor. Instead, both stream processors are immediately reconfigured once a failure has been detected. Therefore it is sufficient to only use an extended Petri net no. 1 as can be seen in Figure 34. As before, the token represents the health status of the system. Because a comparator cannot determine which one of the two redundant stream processors is faulty in case of a mismatch, the token represents both processors. If one of the processors becomes faulty, i.e. one of the failure modes F_x occurs, the token moves via the corresponding exponential transition `mod_seu_*` to place `MOD_F`. After a short failure detection time, modelled by exponential transition `mod_detect`, both stream processors are repaired and the system is up and running again. The repair time, modelled by deterministic transition `mod_rep`, must be the time span necessary to reconfigure *both* stream processors. On the other hand, since a token represents two stream processors, the time values for the exponential transitions

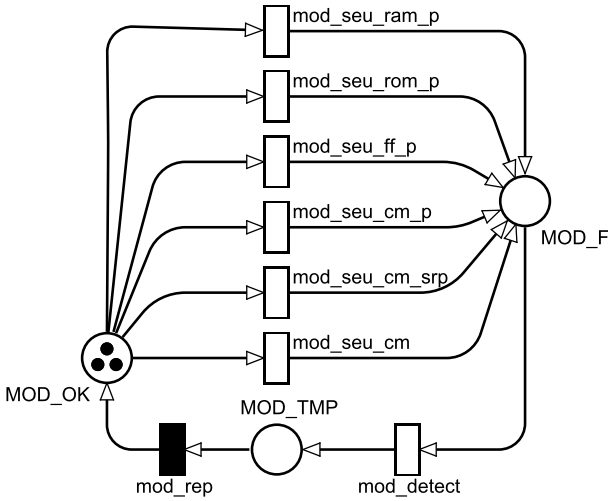

Figure 35: Stochastic Petri net no. 1 (approach 4)

`mod_seu_*` must be halved because the probability of a failure is doubled. Ultimately, the steady state availability is determined by calculating the probability that the token is on place `MOD_OK`.

APPROACH 4 - TRIPLE MODULAR REDUNDANCY If three stream processors are running in hot redundancy, a majority voter can be used for detecting and masking failures, i.e. at least two processors must be faulty at the same time to make the system unavailable. However, if just one stream processor fails, it can be repaired in the background without affecting the operation of the other two processors. To model this FDIR configuration, the Petri net depicted in Figure 35 can be used. It is identical to the one for the Duplication with Comparison configuration except that three tokens are used instead of one. This time, each token represents one stream processor. As a consequence, the exponential transitions `mod_seu_*` are triggered by the MTBF values for just one single stream processor and the repair time `mod_rep` is only the time span required to repair one processor. If one processor fails, one of the three tokens moves from place `MOD_OK` to place `MOD_F` and then back to `MOD_OK` after the detection and repair time elapsed. Ultimately, the steady state availability is determined by calculating the probability that at least two tokens are on place `MOD_OK`.

4.5 CONCLUSIONS

In this chapter, a novel methodology for a systematic availability analysis of stream processors, implemented on SRAM-based FPGAs, has been presented. The proposed methodology provides a new capability allowing the detailed analysis of a circuit without a detailed knowledge of its internal structure. The methodology is implemented as a set of tools integrated together. A new configuration memory failure classification algorithm is proposed, which advances fault injection systems described in literature (see also Section 2.6.2). In addition, a novel Block RAM profiling tool is developed and incorporated allowing Block RAM failure quantification. The proposed approach to availability analysis could easily be automated and is non-intrusive, i.e. it does not require any circuit modifications. Hence, the methodology can serve as an essential tool during the design and qualification phase of space electronics systems that incorporate SRAM-based FPGAs, simplifying the evaluation of the appropriate FDIR mechanisms. This is the first implementation of an ESA standard availability approach with regards to SRAM-based FPGAs, which addresses a gap in the current space qualification process.

5

CHAPTER 5: DEMONSTRATION SYSTEM DESIGN

5.1 INTRODUCTION

In the previous chapters, the theory behind the two main PhD proposals was described. To prove that the theory is true, a proof of concept system is developed, which is now described in detail in this chapter. The system is built from hardware, embedded software and workstation software components. It allows the validation of both the Distributed Failure Detection and the Availability Analysis method.

The chapter is structured as follows. In Section 5.2, the hardware platform and all surrounding test bench components are described. Then, Section 5.3 goes into detail about the implementation of the hardware components of the system, including the stream processor, the FDIR routing switch, the SRAM-based FPGA design and the FDIR supervisor SoC design. Section 5.4 covers the software components, including the embedded FDIR supervisor software, the embedded instrument simulator software, the host PC software and the Block RAM profiling software. Finally, Section 5.5 concludes the chapter.

5.2 HARDWARE PLATFORM

As discussed in Section 2.9, the DRPM system, developed by University of Braunschweig and Airbus Defence and Space, was chosen as hardware platform for the proof of concept implementation. A photo of the DRPM motherboard is shown in Figure 36 and a more detailed view of one of its daughterboards in Figure 37.

The DRPM motherboard comprises a ProASIC3e flash-based FPGA device by Microsemi [139], which implements a SoC that is later used to host the FDIR supervisor application. This SoC is connected to various external memories and several communication interfaces, including SpaceWire, CAN bus and RS-232.

The motherboard is connected to two daughterboards, both comprising SRAM-based FPGAs of type Virtex-4 by Xilinx. Printed Circuit Board (PCB) interconnects enable the communication between the Virtex FPGAs and the ProASIC3e FPGA. Furthermore, the Pro-

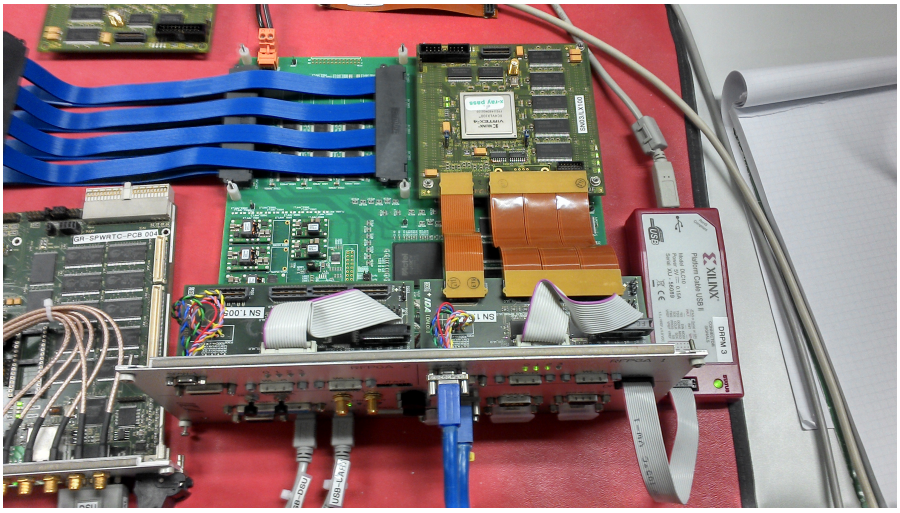

Figure 36: Photo of the DRPM motherboard

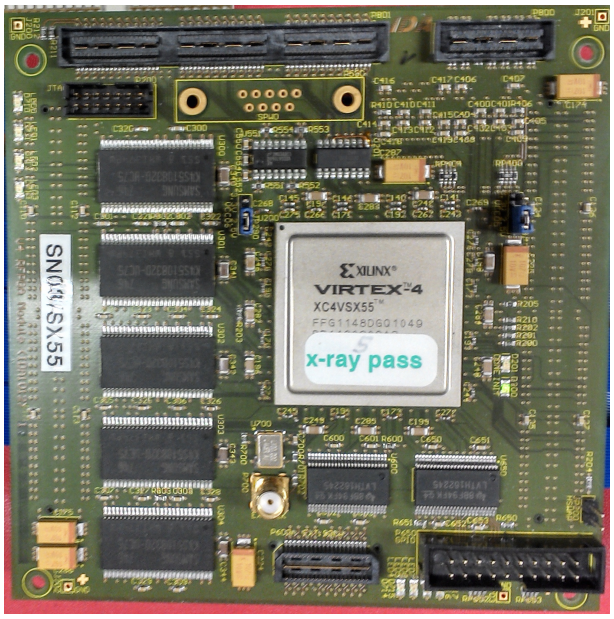

Figure 37: Photo of one of the DRPM daughterboards

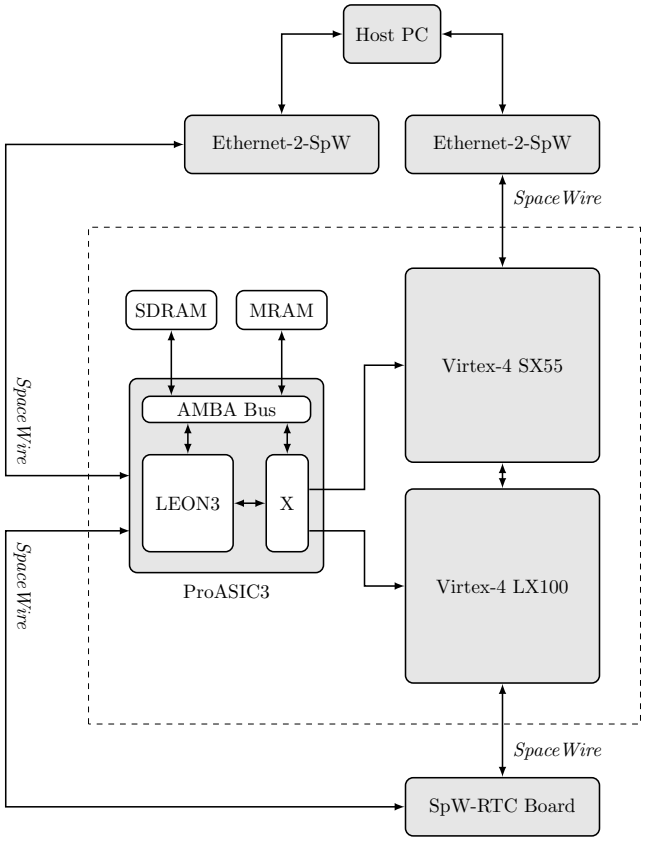

Figure 38: Block diagram of the test bench

ASIC3e device can access the 8-bit wide SelectMAP interfaces of the Virtex-4 FPGAs to (re)configure the devices. In addition, a set of PCB traces enable the intercommunication between the two Virtex devices. On each daughterboard, the Virtex-4 FPGAs are connected to their own SpaceWire interfaces, i.e. direct data communication with stream processors implemented on these devices is feasible.

The DRPM system is integrated into a test bench as shown in the block diagram in Figure 38. The host PC is connected to the DRPM system via two Ethernet-to-SpaceWire bridges. One SpaceWire bridge allows the communication between the PC and the LEON3 microprocessor. The other SpaceWire bridge is used for direct communication with the Virtex-4 FPGAs.

In addition, a SPWRTC development board by Cobham Gaisler [140] is used as data generator and/or data sink. The board is well suited for this task as it comprises a LEON2 microprocessor and two SpaceWire interfaces that can be used in Direct Memory Access (DMA) mode, i.e. data can be transmitted/received directly from/to memory. One of the SpaceWire ports is connected to the SRAM-based

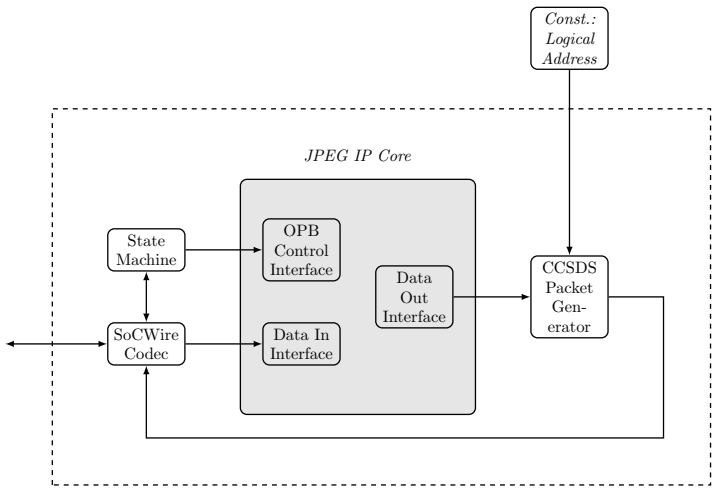

Figure 39: Block diagram of image compression processor

FPGAs. Therefore, data can directly be sent into a processing chain of stream processors. The other SpaceWire port is used for the reception of commands or the transmission of status updates. To save an additional external SpaceWire routing switch, this port is connected to the DRPM system. If the host PC wants to communicate with the SPWRTC board, it can send commands to the DRPM, which then forwards these commands to the SPWRTC board. Similarly, status messages from the SPWRTC board can be sent to the host PC through the DRPM system.

5.3 HARDWARE COMPONENTS

5.3.1 *Network on Chip*

The NoC used for the proof of concept implementation is called SoCWire and was developed by University of Braunschweig. Detailed information can be found in related literature listed in Section 2.7.2. The NoC is well suited for the proof of concept implementation as it fulfills the requirements described in Section 3.3.

5.3.2 *Stream Processor*

It was found that image compression is a good example for typical satellite payload data processing applications. Therefore, an image compression stream processor was implemented as part of the proof

of concept system, which is based on a publicly available Joint Photographic Experts Group (JPEG) IP core from opencores.org [141]. The core is embedded into a stream processor module as depicted in Figure 39. Data enters and leaves the processor via a SoCWire network interface. Input data is expected as raw Red, Green and Blue (RGB) image data in raster scan order with a line width of 640 pixels and a height of 480 pixels.

PROTOCOL Communication with the stream processor is done using Consultative Committee for Space Data Systems (CCSDS) telemetry packets [142] with a maximum payload size of 65,536 bytes. The packet format is defined as follows:

- *Packet Version Number* (3 bits):
  Always zero.
- *Packet Type* (1 bit):
  Always 0 (stream data).
- *Secondary Header Flag* (1 bit):
  Always zero (no secondary header used).
- *Application Process Identifier* (11 bits):
  Not used (but could be used to identify different processors).
- *Grouping Flags* (2 bits):
  01 - first packet, 00 - continuing packet, 10 - last packet.
- *Sequence Count* (14 bits):
  Continuous sequence count, modulo 16384.
- *Packet Data Length* (16 bits):
  Size of the packet payload data - 1.
- *Packet Payload Data* (1 - 65,536 bytes):
  Raw image data as input, compressed JPEG data as output.

STATE MACHINE The stream processor is able to process incoming data streams automatically. The required logic is a state machine as shown in Figure 40. On the one hand, the state machine interacts with the NoC interface and parses the aforementioned CCSDS protocol. On the other hand, it communicates with the embedded JPEG core via a so-called On-chip Peripheral Bus (OPB) interface.

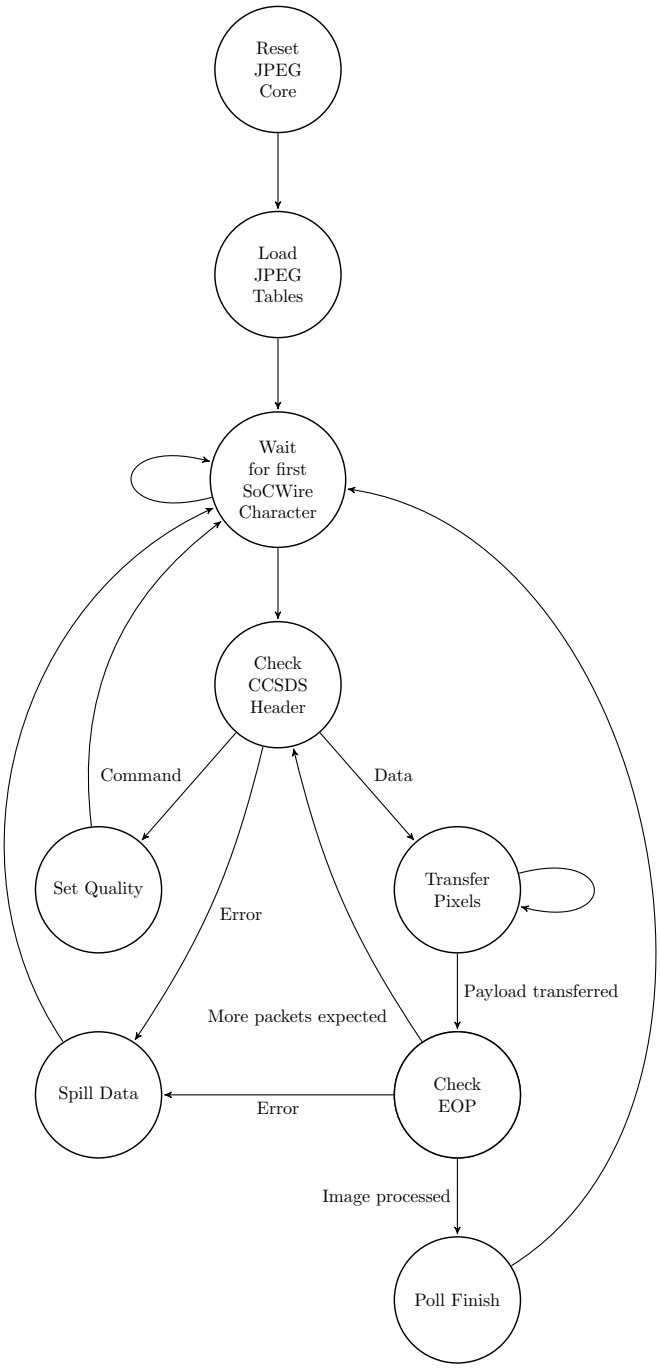

Figure 40: Simplified state machine diagram of the JPEG processor

After an initial configuration of the JPEG core, the state machine waits for the first data character to arrive at the NoC interface. This first data character is the logical address and is therefore skipped. Then, the state machine checks for the correctness of the CCSDS packet header. From this header, the size of the packet payload is extracted. If the received packet contains a command, it is executed (compression level setting) and the state machine waits for the next packet to arrive. If the received packet is a data packet, however, the payload data is shifted into the JPEG core pixel-wise, i.e. byte-wise. Once the whole payload of the packet is shifted in, it is checked to see if the packet is properly terminated with an EOP character. If this is the case, the state machine checks that the image was entirely processed. If so, the state machine waits until the JPEG core is finished with image processing. If not, the state machine waits for the next incoming CCSDS packet.

If an error occurs during protocol parsing, the state machine moves to state *Spill data*. While being in this state, all incoming data is deleted from the receive buffer up to and including the next EOP.

CCSDS PROTOCOL GENERATOR The output of the JPEG core is connected to a protocol generator via a FIFO buffer. Once the FIFO buffer has at least 1,024 bytes available, the state machine of the protocol generator creates a CCSDS packet in the same format as described above, i.e. compressed images are transmitted in 1 kB-sized chunks. The logical network address used for packet transmission is not hard-coded. Instead, the value is taken from an external signal. As mentioned in Section 3.5.3, the reconfigurable partition gets a fixed logical address assigned and not the hosted stream processor. Technically, this is implemented by a constant value in the static area, which is propagated to the packet generator within the stream processor.

5.3.3 *FDIR Routing Switch*

The custom-designed FDIR routing switch is based on the SoCWire routing switch implementation developed by University of Braunschweig. However, as can be seen in the block diagram in Figure 41, it was greatly extended by the following components: (i) a routing table for logical addresses, (ii) a voter module, (iii) a multicast mechanism and (iv) a configuration port, which allows the programming of the

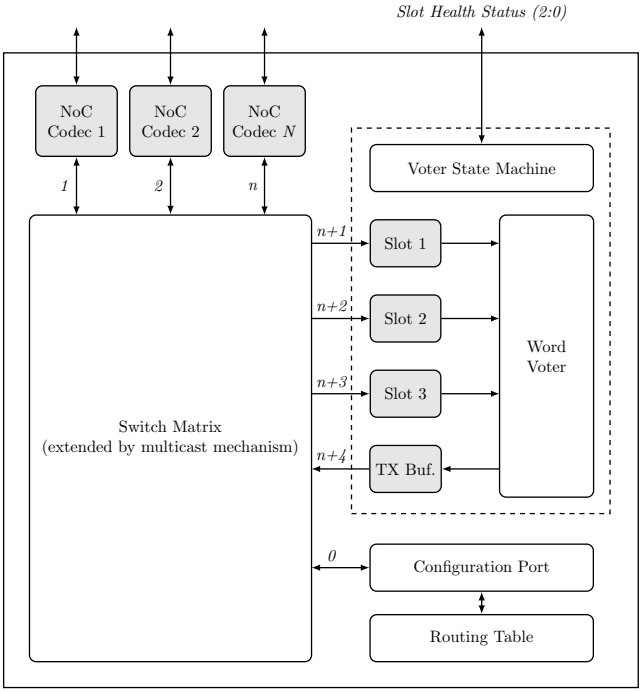

Figure 41: Block diagram of FDIR routing switch

routing table and the timeout values that are needed for the voter module and the multicast mechanism.

ROUTING TABLE / TIMEOUT VALUES The routing table is implemented in a 32x32 bit Dual-Port RAM within the configuration port, i.e. up to 32 logical addresses are supported. The 32-bit word for each logical address is divided into two halves: in the lower 16 bits, each bit represents an output port to which the logical address is assigned. For instance, if a packet arrives with a logical address that is assigned to `0x0e`, the packet will be multicast to port 1, 2 and 3. In the upper 16 bits, each bit represents an input port at which packets are allowed to arrive with this particular logical address, see also Section 3.5.3. For instance, a value of `0x0f` means that packets with this logical address are only permitted to arrive at port 0, 1, 2 and 3.

If a new packet arrives at one of the input ports, a so-called entrance logic extracts the logical address from the packet. Then, it will request the lower bit vector from the routing table for this particular address. Since one entrance logic exists for each input port, the access to the routing table must be done via a fair round robin arbitration: First, the entrance logic puts the desired logical address on a request bus. Then, once the routing table has the bit vector for this address available, it

asserts a valid signal so the entrance logic can continue to process the new packet. At the same time, the entrance logic is connected via a single signal to the corresponding bit position in the upper bit vector. Hence, the entrance logic is immediately notified if the new packet is actually allowed to be processed at its input port. If not, the entrance logic will delete the complete packet up to and including the next EOP character.

Within the configuration port, additional registers store the multicast, inter-packet, inter-character and last-resort timeout value. All values are expressed in clock cycles and stored in 24-bit wide registers. Since the Virtex-4 FPGAs run at 100 MHz, the maximum timeout span is approximately 168 ms.

CONFIGURATION PORT As can be seen in Figure 41, the configuration port is connected to the switch matrix via physical port number 0. Aside from hosting the aforementioned routing table and timeout value registers, it also comprises a state machine that can receive and transmit configuration packets in CCSDS packet format:

- *Packet Version Number* (3 bits):
  Always zero.
- *Packet Type* (1 bit):
  Always 1 (configuration command).
- *Secondary Header Flag* (1 bit):
  Always zero.
- *Application Process Identifier* (11 bits):
  Not used.
- *Grouping Flags* (2 bits):
  11 - unsegmented data.
- *Sequence Count* (14 bits):
  Used as command identifier (lower 8 bits): `0x01` - request routing table, `0x02` - request timeout values, `0x03` - set routing table, `0x04` - set timeout values.
- *Packet Data Length* (16 bits):
  Size of the packet payload data - 1.
- *Packet Payload Data* (1 - 65,536 bytes):
  Command `0x01` and `0x02`: One dummy octet. Command `0x03`:

128 octets with routing table values is ascending order, i.e. from logical address 32 to 63, MSB first. Command `0x04`: 12 octets with (i) broadcast timeout, (ii) inter-packet timeout, (iii) inter-character timeout and (iv) last-resort timeout, MSB first.

If a command is sent to the configuration port to retrieve either the routing table or the timeout values, the configuration port will send out the data in the same format as described above. The network packet will be transmitted using the logical address of the configuration port. Thus, it is first required to assign this logical address to a physical port by programming the routing table. Retrieving the values back from the routing switch is a simple way of ensuring that the data was transmitted and stored correctly before.

VOTER MODULE The voter module was designed and integrated into the routing switch as proposed in Section 3.5.1. The pseudo-code given in that section is nearly identical to the real HDL implementation and a separate description of its functionality is therefore not necessary. As can be seen in Figure 41, the slots of the voter module (dashed box) are connected to the switch matrix via port $n+1$, $n+2$ and $n+3$. Thus, the real port number of each slot depends on the total number of physical ports available in the routing switch, which can be configured during synthesis with a generic value. The output of the voter module is connected to the switch matrix too, allowing full flexibility in routing the network traffic. Any input port of the routing switch can be routed to any slot of the voter module and the output of the voter can leave the routing switch via any output port.

MULTICAST MECHANISM The concept of the multicast mechanism was briefly discussed in Section 3.5.2. To make it part of the routing switch, however, it must be incorporated into the switch matrix. Then, the number of output ports is flexible and it is not predefined which output ports of the matrix are actually used for the multicast (as this is defined by the logical address mapping, see Section 5.3.3 above).

The combinational data multiplexing logic of the switch matrix was extended as follows:

- Check for every input port if output port(s) are currently connected and if multicasting is currently allowed for these output port(s).

- If this is the case, find all connected output ports, count their number and check how many of them are full, i.e. how many are blocking outgoing traffic.
- If one and only one output port is blocking but more than one output ports are connected, a multicast timeout register reload signal for this input port is *not* asserted, i.e. the timer is active.
- If several output ports are connected to one input port, no data is read from the input port if at least one output port is blocked.
- Similarly, no data is written to the output ports if at least one output port is blocked.

A multicast timeout register exists for each input port of the routing switch. A dedicated sequential logic keeps track of the current state of these registers:

- If the data multiplexing logic asserts a timeout register reload signal, the corresponding counter is reloaded. Otherwise, it is decremented every clock cycle.
- If the timeout register of a specific input port reaches zero, the logic checks which connected output port is blocking. Then, the blocking output port is removed from the multicast round by setting a corresponding disable flag.
- Once the data transfer between an input port and some output ports is finished, the above mentioned disable flags are reset, i.e. an output port can only be removed from the multicast mechanism for the duration of one network packet. With the next packet, the routing switch will try to use this output port again. This is important because if a faulty, blocking stream processor is repaired, it must automatically be supplied with new input data afterwards.

5.3.4 *Virtex-4 FPGA Design*

A block diagram of the Virtex-4 FPGA design is depicted in Figure 42. It comprises: (i) three reconfigurable partitions, all connected to the (ii) FDIR routing switch described in Section 5.3.3 above, (iii) a SpaceWire interface, (iv) a SpaceWire to NoC bridge, allowing the communication between the NoC and external components via SpaceWire

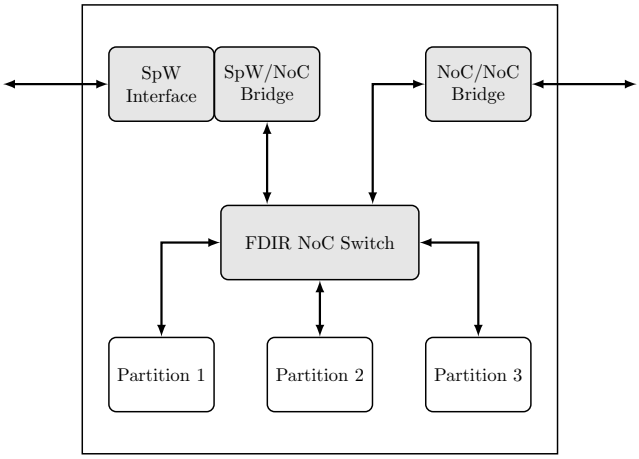

Figure 42: Block diagram of Virtex-4 FPGA design

and (v) a NoC to NoC bridge, allowing the communication between the first and the second Virtex-4 FPGA.

RECONFIGURABLE PARTITIONS Each partition is large enough to host a JPEG image processing stream processor. To implement the FPGA design, the methodology described in the Xilinx Partial Reconfiguration User Guide [143] was followed. In first experiments, the workflow based on the PlanAhead tool was used, which greatly simplifies the process of finding the right dimensions for each partition due to its simple graphical user interface. A screenshot of the final FPGA floorplan, extracted from this tool, is shown in Figure 43. The reconfigurable partitions are shown as white boxes, whereas the so-called static area is shown in blue. All NoC and FDIR components are implemented in this static area.

Later on it was found that the script-based workflow is much more convenient, faster and reliable. First, the JPEG stream processor as well as the static area is pre-synthesised using for instance the XST synthesis tool. This process must be done just one time. Then, a Tcl script maps, places and routes the netlist of each stream processor using the Xilinx toolchain. The main output of this workflow is a full bitstream comprising the static area with a JPEG stream processor preplaced on each partition and a partial bitstream for each JPEG stream processor, which can later be used for failure recovery. In addition, blank bitstreams are created for each partition, which can be used to erase a stream processor, e.g. to save power. The size of each

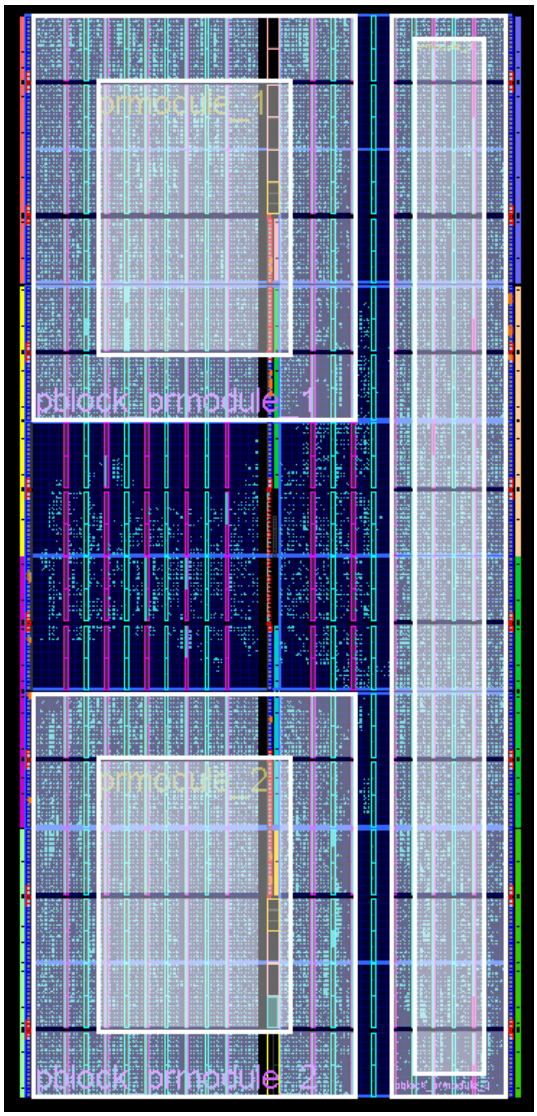

Figure 43: Virtex-4 FPGA floorplan

partition is defined by area constraints in the UCF constraint file, an example for the first partition is shown in Listing 10.

Listing 10: Area constraints for partition 1

```
INST "prmodule_1" AREA_GROUP = "pblock_prmodule_1";
AREA_GROUP "pblock_prmodule_1" RANGE=SLICE_X0Y160:SLICE_X63Y255;
AREA_GROUP "pblock_prmodule_1" RANGE=DSP48_X0Y40:DSP48_X4Y63;
AREA_GROUP "pblock_prmodule_1" RANGE=RAMB16_X0Y20:RAMB16_X6Y31;
```

As can be seen, the dimensions must be defined for each type of building block used in the design. For the JPEG stream processor, it is sufficient to define areas for CLB slices, DSP blocks and Block RAMs.

SPACEWIRE INTERFACE Although not fully related to the work on this PhD thesis, a SpaceWire interface was developed as a side project, replacing the original SpaceWire interface by STAR-Dundee [144], which was delivered by University of Braunschweig as part of the DRPM system. In contrast to the original interface, the custom-built SpaceWire interface is fully synchronous. Instead of doing asynchronous clock recovery, the incoming data-strobe signals are oversampled. As a consequence, not one single timing constraint is required, a fact that prevented some frustration during the FPGA design phase.

The design of the interface is rather simple as it is fixed to one clock frequency. Using double data rate output registers and four times input data oversampling, data can be transmitted and received with a data rate of 200 Mbit/s at 100 MHz clock frequency.

SPACEWIRE TO NOC BRIDGE This bridge is basically a SoCWire core whose application-side interface is connected to the application-side interface of the SpaceWire core. Both network components use a FIFO-like interface, allowing a simple handshake between both. The inverted full flag of the first core can directly be connected to the read enable signal of the second core, whereas the inverted empty flag of the second core can be connected to the write enable signal of the first core.

NOC TO NOC BRIDGE Similarly to the SpaceWire to NoC bridge, the NoC to NoC bridge connects the application-side interface of one SoCWire core to the application-side interface of another SoCWire

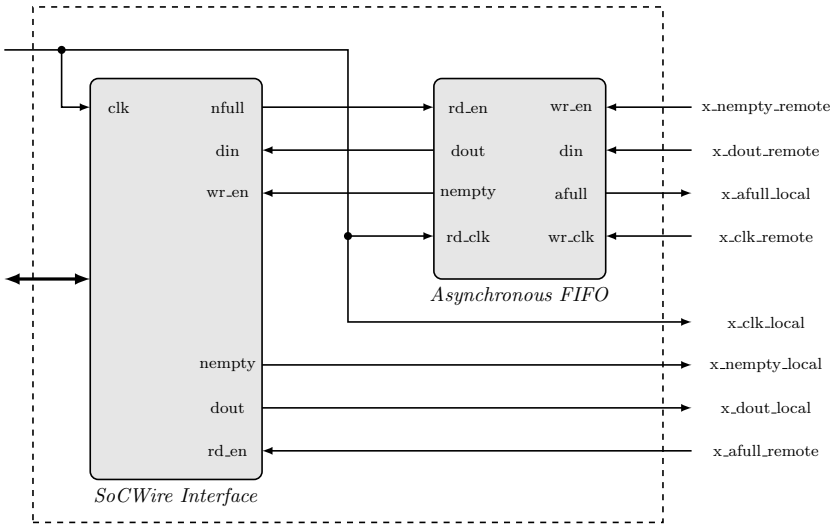

Figure 44: Block diagram of NoC to NoC bridge

core. However, while the first core is implemented on the local FPGA, the second core is implemented on the remote FPGA. As a consequence, the data, read and write signals are bridged from one Virtex-4 FPGA to the other Virtex-4 FPGA via PCB traces and are therefore crossing clock domains.

To avoid unreliable data communication due to metastability and setup and hold violations, an asynchronous FIFO buffer is used for transferring the data from one clock domain to the other. Figure 44 shows a block diagram of the NoC to NoC bridge, comprising the SoCWire interface and the FIFO buffer.

Both the read side of the FIFO buffer and the SoCWire core are clocked by the local clock. Signal `dout` of the FIFO buffer is connected to `din` of the SoCWire core, the inverted full signal of the SoCWire core `nfull` is used as read enable signal `rd_en` on the FIFO side and the inverted empty signal of the FIFO buffer `nempty` is used as write enable signal `wr_en` on the SoCWire side. The output side of the SoCWire core is directly connected to the remote FPGA via `x_nempty_local` and `x_dout_local`. For handshaking, an almost full flag `x_afull_remote` is received from the remote side.

On the write side of the FIFO, the `wr_en` and `din` signals are provided from the remote FPGA and are sampled using the remote clock `x_clk_remote`. For handshaking, an almost full flag `x_afull_local` is transmitted to the remote side.

To further improve reliability, all signals provided by the remote FPGA are sampled in IOB flip-flops. The flip-flops are clocked by the

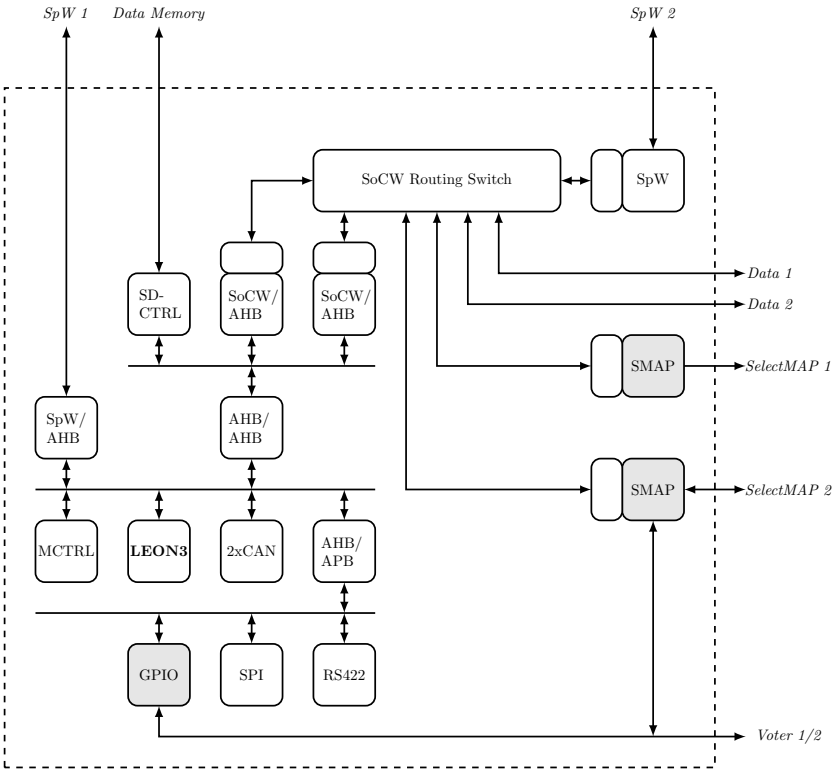

Figure 45: Block diagram of FDIR supervisor SoC

remote clock that is first recovered by a source-synchronous Digital Clock Manager (DCM). In the other direction, the local almost full flag `x_afull_local` is also sampled in an IOB flip-flop using the recovered remote clock.

On transmission side, the outgoing signals are sampled in IOB flip-flops using the local clock. A replica of the local clock is created in an IOB too, using an output double data rate register. On rising edge a constant value of 0 is registered, on falling edge a constant value of 1 (inverted clock).

By sampling the data already in the IOBs and by replicating the clocks, possible skew between the data and clock lines is limited to the PCB traces. Since the propagation delays of the PCB traces are not known, the inverted clock is transmitted, which gives some additional room for skew.

5.3.5 *FDIR Supervisor SoC Design*

A block diagram of the SoC design, which is implemented on the flash-based ProASIC3e FPGA, is shown in Figure 45. The rather com-

plex SoC was developed by University of Braunschweig as part of the DRPM system. The SoC is based on the GRLIB by Cobham Gaisler [145] and includes a LEON3 microprocessor, which is used to run the FDIR supervisor software, see Section 5.4.1.

Since the SoC proved to be well designed, only a handful of modifications were done (highlighted in grey): (i) an additional General Purpose Input Output (GPIO) port was added to the AMBA processor bus to communicate with the voter modules inside the FDIR routing switches and (ii) a SelectMAP (SMAP) core was redesigned from scratch to also enable bitstream readbacks from the Virtex-4 devices.

ADDITIONAL GPIO PORT The GPIO port is 16-bit wide, divided in two halves. The lower eight pins are used to interface the voter module on the first FPGA, the upper eight pins are used to interface the voter module on the second FPGA. The lower four pins of each half are used by the voter module to transmit a debug code to the FDIR supervisor, which is defined as follows:

- `000`: No error.
- `001`: Inter-packet timeout occurred.
- `010`: Inter-character timeout occurred.
- `011`: Voting error occurred.
- `100`: Last-resort timeout occurred.

Three of the four upper pins of each half are both input and output pins, whereas the fourth bit is used to switch the direction of these pins. If the pins are configured as input, the FDIR supervisor software can retrieve the current slot status from the voter module, e.g. the value `011` means that slots 1 and 2 are healthy whereas slot 3 is marked as faulty. If the pins are configured as outputs, the FDIR supervisor can update the slot status of the voter module after failure recovery.

The GPIO port triggers an interrupt once one of the signals changes. The FDIR supervisor software can catch this interrupt in a service routine to initiate an appropriate failure recovery action.

SELECTMAP CORE The redesigned SelectMAP cores are used to interface the 8-bit wide SelectMAP interfaces of the Virtex-4 FPGAs. Goal of the redesign was the possibility to also read back bitstreams

from the Virtex-4 devices and to send them to any component in the SoC, e.g. to the LEON3 microprocessor or to the SpaceWire interface.

The original functionality remains: A full or partial bitstream can be sent through the SoCWire network to one of the SelectMAP cores. Since the bitstream includes all commands necessary to interface the configuration state machine of the Virtex-4 FPGAs, it can simply be clocked into the device without any modifications. Due to the nature of the SoC design, the bitstream can either be sent from the LEON3 processor or transferred directly from the data memory via DMA.

Reading back the bitstream is slightly more complex: First, the SelectMAP interface must be switched to write mode. Then, the commands in Listing 11 are written to the interface.

Listing 11: SelectMAP commands for bitstream readback

```
x"aa", x"99", x"55", x"66",         -- sync word
x"20", x"00", x"00", x"00",         -- noop
x"30", x"00", x"80", x"01",         -- rcrc
x"00", x"00", x"00", x"07",
x"20", x"00", x"00", x"00",         -- noop
x"20", x"00", x"00", x"00",         -- noop
x"30", x"00", x"80", x"01",         -- shutdown
x"00", x"00", x"00", x"0b",
x"20", x"00", x"00", x"00",         -- noop
x"30", x"00", x"80", x"01",         -- rcrc
x"00", x"00", x"00", x"07",
x"20", x"00", x"00", x"00",         -- noop
x"20", x"00", x"00", x"00",         -- noop
x"20", x"00", x"00", x"00",         -- noop
x"20", x"00", x"00", x"00",         -- noop
x"20", x"00", x"00", x"00",         -- noop
x"20", x"00", x"00", x"00",         -- noop
x"30", x"00", x"80", x"01",         -- rcfg
x"00", x"00", x"00", x"04",
x"30", x"00", x"20", x"01",         -- far: start from 0x00
x"00", x"00", x"00", x"00",
x"28", x"00", x"60", x"00",         -- read from fdro
x"40", x"0a", x"d3", x"81",         -- 0xad381 = 709,505
                                    -- (all frames from
                                    -- sx55 + one dummy)
x"20", x"00", x"00", x"00",         -- noop
x"20", x"00", x"00", x"00";         -- noop
```

After synchronising the interface, a shutdown command is sent to the Virtex-4 FPGA. Then, the FAR is set to 0, i.e. the full bitstream is read back from the first byte position. By writing the desired number of frames (`0xad381`) to the device, the readback is activated.

Now, the SelectMAP interface must be switched to read mode. The bitstream is clocked out from the Virtex-4 device byte by byte and stored in a FIFO buffer, which is connected on the read side to an internal SoCWire interface. Once all bytes are read, the SelectMAP interface is switched back to write mode and the commands in Listing 12 are written.

Listing 12: SelectMAP commands for restarting the FPGA

```
x"20", x"00", x"00", x"00",         -- noop
x"30", x"00", x"80", x"01",         -- start
x"00", x"00", x"00", x"05",
x"30", x"00", x"80", x"01",         -- rcrc
x"00", x"00", x"00", x"07",
x"30", x"00", x"80", x"01",         -- desync
x"00", x"00", x"00", x"0d",
x"20", x"00", x"00", x"00",         -- noop
x"20", x"00", x"00", x"00";         -- noop
```

As can be seen, the Virtex-4 device is first started, the CRC register is then reset and the SelectMAP interface finally de-synchronised.

5.4 SOFTWARE COMPONENTS

5.4.1 *FDIR Supervisor Software*

Core of the embedded system is the FDIR supervisor software running on the LEON3 microprocessor. Its purpose is two-fold: On the one hand it executes failure recovery actions for faulty stream processors. On the other hand it includes several functions for availability analysis and hardware verification.

The software is based on the real time operating system RTEMS [146] and comprises several tasks as depicted in Figure 46. Inter-task communication is done via message queues and events. In the following, a brief overview of the each task is given.

INITIALISATION TASK Execution of the application begins with the initialisation task. First, the peripherals are initialised, including

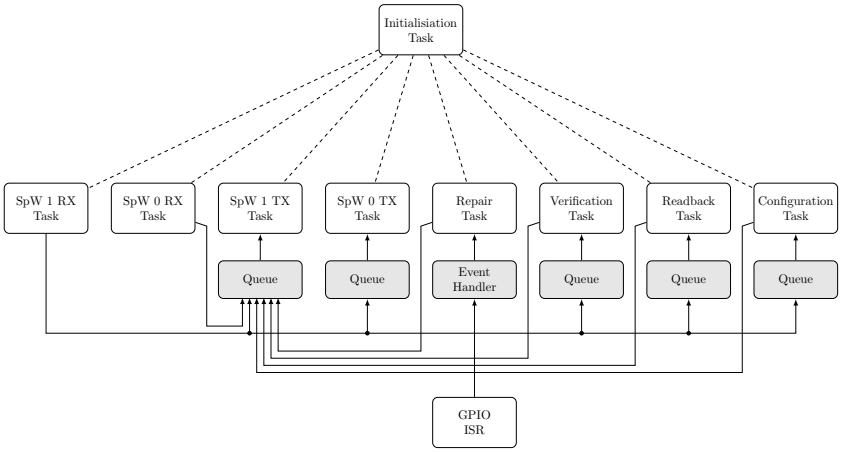

Figure 46: Overview of FDIR supervisor software tasks

system clock, SD-RAM, file system, and all NoC components. Then, the message queues, tasks and semaphores are created. Thereafter, the tasks are started and the pointers to the hardware registers, the interrupt handler and the stream handlers are initialised. Finally, the task deletes itself.

SPACEWIRE 1 RECEIVE TASK This task handles incoming data from the second SpaceWire interface that is connected to the host PC. The following commands are supported:

- `OPEN_FILE`: Opens a file in the file system. If it does not exist, the file is created first.
- `DELETE_ALL_FILES`: All files are deleted from the file system.
- `CLOSE_FILE`: Closes the currently opened file.
- `PRINT_TABLE`: Prints a list of all files in the file system to the serial debug port.
- `GET_FILE_INFO`: Returns the data length and the name of a particular file in the file system via SpaceWire.
- `GET_NO_FILES`: Returns the number of files in the file system via SpaceWire.
- `PRINT_VOTER_STATUS`: Prints the health status of the voter slots to the serial debug port.
- `GET_VOTER_STATUS`: Returns the health status of the voter slots via SpaceWire.

- `RESET_VOTER_STATUS`: Resets the health status of the voters to a specific value.
- `FULL_CONFIG`: Initiates a full configuration (via CPU) with a given full bitstream file. This is done by sending a message to the Configuration Task.
- `PARTIAL_CONFIG`: Initiates a partial reconfiguration (via CPU) with a given partial bitstream file. This is done by sending a message to the Configuration Task.
- `PARTIAL_CONFIG_DMA`: Initiates a partial reconfiguration (via DMA transfer) with a given partial bitstream file. This is done by sending a message to the Configuration Task.
- `FDIR_MAPPING`: Maps partial bitstreams of stream processors to voter slots. This is necessary because the supervisor must be aware of which stream processor output is routed to which voter slot.
- `CREATE_DMA_FILE`: Modifies partial bitstreams in such a way that they can be transferred directly from memory to the SelectMAP cores via SoCWire.
- `FPGA_RESET`: Resets a particular Virtex-4 device or a particular reconfigurable partition on one of the Virtex-4 devices.
- `SET_DEBUG_OUTPUT`: Switches debug output via serial port on/off.
- `INJECT_FAULT`: Takes a desired configuration frame from memory and flips a desired bit within the frame before downloading it to one of the Virtex-4 devices. The same command can be used to scrub a frame.
- `VOTER_VERI`: Starts the voter verification procedure by sending a message to the Verification Task.
- `READBACK`: Triggers a bitstream readback from a Virtex-4 device by sending a message to the Readback Task.
- `FILE_DOWNLOAD`: Returns a requested file from the file system to the host PC via SpaceWire.
- `TRANSFER_ROUTING_TABLE`: Forwards the routing tables for the FDIR routing switches to the SPWRTC board. Then, the instrument simulator software can configure the routing switches with these tables.

- `TRANSFER_TIMEOUTS`: Forwards the timeout values for the voter modules and the multicast mechanisms to the SPWRTC board. Then, the instrument simulator software can configure the routing switches with these values.
- `TRANSFER_IMAGE`: Transfers a raw test image to the SPWRTC board. Beforehand, the raw image must be uploaded to the file system of the DRPM system.
- `SEND_IMAGE`: This command is forwarded to the SPWRTC board.
- `TRANSFER_JPEG`: This command is forwarded to the SPWRTC board.
- `RESET_AVAIL`: This command is forwarded to the SPWRTC board.

Nearly all above listed commands acknowledge their correct execution by transmitting an answer packet to the host PC via SpaceWire. Often, these answer packets contain additional status and debug information. Aside from the listed commands, the task also supports file uploads from the host PC to the file system via SpaceWire.

SPACEWIRE 0 RECEIVE TASK This task handles incoming data from the first SpaceWire interface, which is connected to the SPWRTC board, i.e. it handles status updates transmitted by the instrument simulator software. All updates are directly forwarded to the host PC by sending them to the message queue of the SpaceWire 1 Transmit Task.

SPACEWIRE 0 & 1 TRANSMIT TASKS These two tasks transmit data from their message queues to the corresponding SpaceWire interfaces, i.e. to the host PC or the SPWRTC board.

REPAIR TASK The repair task is triggered by an event generated due to an GPIO interrupt, i.e. when the health status of some voter slot changes. First, the routine checks which slots are marked as faulty and which stream processors are mapped to these slot. Then, it tries to repair these stream processors by means of partial reconfiguration via DMA transfer. Next, the routine updates the health statuses of the corresponding slots and waits until the voter successfully reintegrates the freshly repaired stream processors. Finally, a status information is sent to the host PC, which contains the numbers of the repaired slots as well as the time that was required to repair the system. During

the proton irradiation test campaign, see Chapter 8, the routine also initiated a readback of the partial stream processor bitstream.

VOTER VERIFICATION TASK This task can be used to verify the correct functionality of the voter modules. Beforehand, a database file must be uploaded to the DRPM system, which comprises a list of known to be sensitive configuration memory bits within a particular partial stream processor bitstream.

The algorithm injects a fault into each listed configuration memory bit. After each injection, it waits until the assigned voter module signals a detected failure via its health status. It is checked if the correct slot was marked as faulty and it is also checked that no false-negatives are signaled for the other slots. Finally, the stream processor is repaired by means of partial reconfiguration and the voter health status updated before the next fault is injected.

For each injected fault, a status information is sent to the host PC, which contains the injected bit position, the time that was needed to detect the failure and the time that was needed to recover the failure. The algorithm stops automatically if (i) a wrong slot is detected as being faulty, (ii) the voter module is unable to reintegrate a stream processor after failure recovery or (iii) all sensitive bits in the database file were successfully tested.

READBACK TASK This task is able to trigger a bitstream readback by communicating with the SelectMAP core described in Section 5.3.5. The communication is partly done by sending messages via the NoC and partly via a dedicated GPIO port. The readback of full bitstreams and the readback of single configuration frames is supported.

First, the SelectMAP core is set up with so-called forward addresses, which either routes the bitstream to the LEON3 microprocessor or the SpaceWire 1 interface, i.e. directly back to the host PC. For performance reasons, the latter approach was chosen during the proton irradiation test campaign. Then, the SelectMAP core is configured to either read the full bitstream or a single frame. In case of a single frame, also the FAR address of the frame is configured. All aforementioned configuration steps are done by sending appropriate messages through the NoC to the SelectMAP core.

Finally, the readback is triggered via a dedicated GPIO pin. Once the readback is active, the task polls another GPIO pin to determine when the readback is finished.

During the design phase, a small bug in the SoCWire Protocol (SoCP) core was found. SoCP is the mailbox protocol for SoCWire that was also developed by University of Braunschweig [129]. It is used on top of SoCWire and allows the reading and writing of registers within the core. This functionality is utilised by the aforementioned configuration steps. At the same time, the SoCP core also allows data streaming, which is used for the transmission of the bitstream. It turned out that streams are likely to get corrupted when a register is written or read during their transmission, making a polling of a register impossible while a readback is active. Hence, the NoC was only used for the initial configuration, whereas the polling was realised via an additional GPIO port.

CONFIGURATION TASK This task can reconfigure a Virtex-4 FPGA with either a partial or a full bitstream. Two methods are implemented: The simple but slow method reads the bitstream block-wise from the SD-RAM into the main memory of the CPU. Then, each block is sent via the NoC to the SelectMAP core, which is transferring the data to the SelectMAP interface of the Virtex-4 FPGA. The more complex but much quicker method is using a DMA transfer, i.e. the bitstream is directly sent from the SD-RAM controller to the SelectMAP interface. Beforehand, however, the bitstreams must be prepared using the `CREATE_DMA_FILE` command. After transfer, the task checks if the (re)configuration was successful and sends an appropriate status information to the host PC.

5.4.2 *Instrument Simulator Software*

As mentioned in Section 5.2, the test bench of the proof of concept system comprises a LEON2-based SPWRTC board. The microprocessor on this board runs a software written in Bare-C that acts primarily as an instrument simulator, which is described in detail hereafter. The following functions are implemented:

- The software is used to transmit raw test images to the stream processors with a frequency of 10 Frames Per Second (FPS).
- The software also receives the returned JPEG images and compares them to a pre-stored golden copy. Thus, the availability can be measured by counting the number of transmitted raw images and the number of correctly received JPEG images.

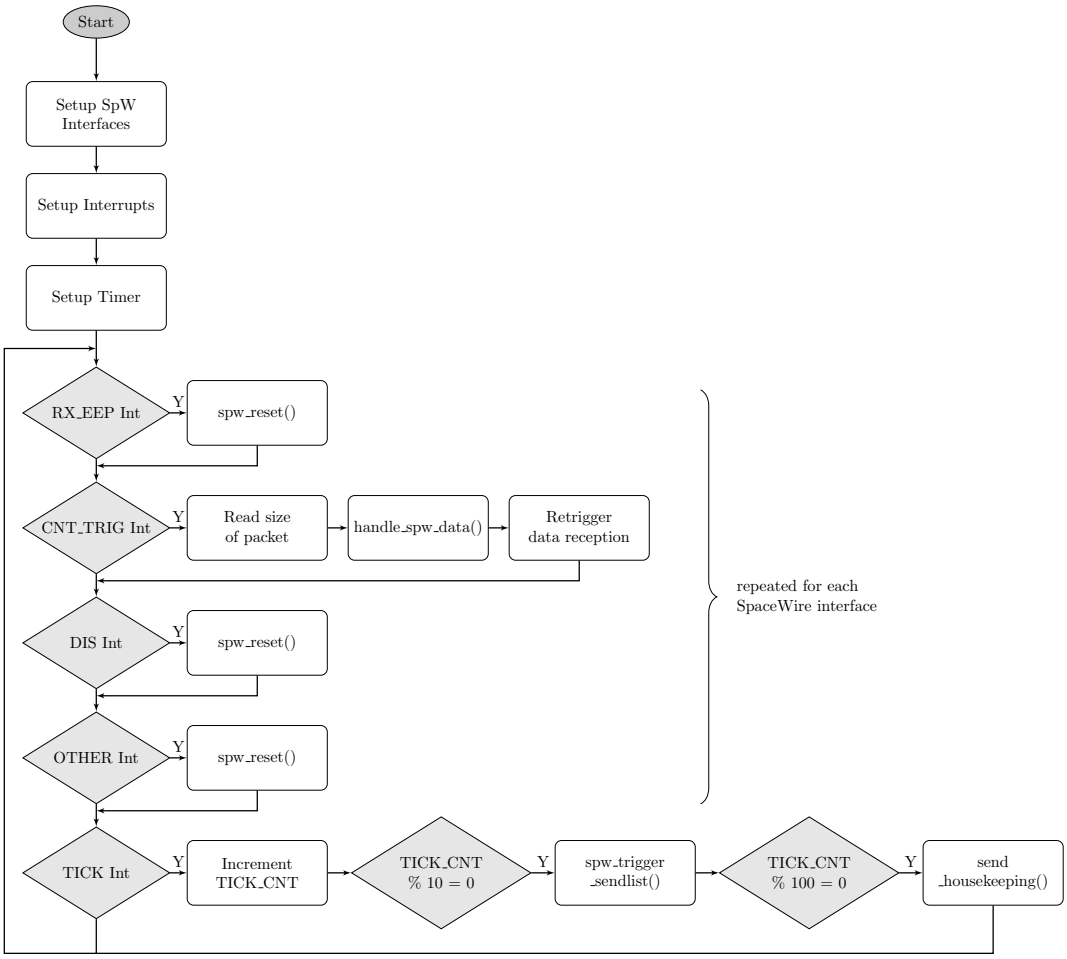

Figure 47: Flowchart of instrument simulator software main function

- Since the SPWRTC board is directly connected to the Virtex-4 FPGAs, the software can also be used to transmit routing tables and timeout values to the FDIR routing switches.
- In fixed intervals, status information is transmitted to the host PC via the DRPM system, including the current time stamp and the number of transmitted and correctly received images.

MAIN() FUNCTION A flowchart of the `main()` function is shown in Figure 47. Before the function enters the infinite while loop, the SpaceWire interfaces, the interrupts and the timer are set up. SpaceWire interface 0, connected to the DRPM, is set up to run at 50 Mbit/s and SpaceWire interface 1, connected to the Virtex-4 FPGAs, is set up to run at 200 Mbit/s. For both interfaces, interrupts are enabled that indicate if the link received an Error End of Packet (EEP) character (`RX_EEP`) or got disconnected (`DIS`). Another interrupt `CNT_TRIG` is triggered once a full SpaceWire packet has been received. Finally, the timer is set up to trigger a `TICK` interrupt every 10 ms.

The software implements service routines, which set appropriate flags once an interrupt occurs. These flags are polled within the infinite while loop for each SpaceWire interface. If an `RX_EEP` or `DIS` or some other unknown interrupt occurs, the SpaceWire interface is reset, i.e. it is first disconnected, then reconnected and finally the reception of the next SpaceWire packet is enabled. This failure recovery approach is required to handle all kinds of corrupted network streams, which might be produced by faulty stream processors, e.g. during a fault injection campaign. If a `CNT_TRIG` interrupt occurs, the size of the received packet is determined and the function `handle_spw[0|1]_data()` is called. Then, the data reception for the next packet is enabled. If a `TICK` interrupt occurs, a counter is incremented, which is used as divider for the tick signal. Every 100 ms, a function `spw_trigger_sendlist()` is called. Every second, a function `send_housekeeping()` is called, which transmits the earlier mentioned status information to the host PC via the DRPM system.

HANDLE_SPW0_DATA() FUNCTION This function handles data received at SpaceWire interface 0, connected to the DRPM system. Similarly to the FDIR supervisor software, a couple of commands are supported:

- `TRANSFER_ROUTING_TABLE`: Stores a routing table for the FDIR routing switch to memory. Then, the routing table is transmit-

ted to the Virtex-4 FPGAs via SpaceWire interface 1. Finally, a command is sent to the FDIR routing switch to retrieve the routing table back to verify that the transmission was successful.

- `TRANSFER_TIMEOUTS`: Stores timeout values for the FDIR routing switch to memory. Then, the values are transmitted to the Virtex-4 FPGAs via SpaceWire interface 1. Finally, the values are retrieved back from the switch to verify that the transmission was successful.
- `SEND_IMAGE`: Toggles the periodic transmission of raw images to the stream processors.
- `RESET_AVAIL`: Resets the counters for the transmitted and correctly received images.
- `TRANSFER_JPEG`: Transmits the golden JPEG image via SpaceWire to the DRPM file system. Then, the host PC can retrieve it from the DRPM system.

Aside from the listed commands, the function also supports file uploads from the DRPM system. The file upload capability is used for transferring the raw test image from the host PC via the DRPM to the main memory of the SPWRTC board. Once the full raw image is received, the function packs the data into 15 CCSDS packets and memory pointers to these packets are stored in a so-called *send list*. This send list can be triggered by the instrument simulator software. As a result, the CCSDS packets are automatically transmitted to the Virtex-4 FPGAs via SpaceWire interface 1 using DMA transfer. After setting up the send list, it is triggered one time by calling the `spw_trigger_sendlist()` function to create the initial golden JPEG copy.

HANDLE_SPW1_DATA() FUNCTION This function handles data received at SpaceWire interface 1, connected to the Virtex-4 FPGAs:

- If a JPEG image is received in CCSDS packet format, it is checked for errors, unpacked, stored in memory and finally compared to the golden JPEG copy. If both files are identical, the counter for the correctly received images is incremented. However, if no golden copy is available yet, the received file is stored in memory for this purpose.

- If a CCSDS packet is received from a FDIR routing switch and the packet contains a routing table, the routing table is compared to the one, which was transmitted to the routing switch before. Then, a status information is transmitted to the host PC via the DRPM system indicating if the received table matches the transmitted one.
- Similarly, if timeout values are received back from a FDIR routing switch, these values are compared to the ones, which were transmitted to the routing switch before. Again, a status information is transmitted to the host PC indicating the outcome of the test.

SPW_TRIGGER_SENDLIST() FUNCTIONS These functions are used to trigger the transmission of one or more SpaceWire packets, defined in the send list, via one of the SpaceWire interfaces. Beforehand, however, the send lists must be set up. This is done by first storing the packets in memory. Then, a memory pointer to the data block as well as the size of the block is stored in a structure, which is added to the send list. For instance, the raw image is stored in several packets in memory. By simply setting up and triggering the send list, they are transferred to the SpaceWire interface via DMA.

For convenience, wrapper functions are implemented, which set up and trigger a send list with just one entry. The memory pointer and the block size can be given as function parameters. For example, these functions are used to transmit the routing tables to the Virtex-4 devices or status information packets to the DRPM system.

5.4.3 *Host PC Software*

The software running on the host PC is written in C++ using the platform-independent Qt framework [147]. During development, the software was successfully compiled for Linux, OSX and Windows. It is used for sending commands to and receiving status information from the embedded FDIR supervisor and instrument simulator software via SpaceWire.

The graphical user interface of the software is divided into two halves. In the top half, the controls are arranged in seven tabs, each responsible for a specific task or group of tasks. In the bottom half, status information from both the embedded systems and the host PC

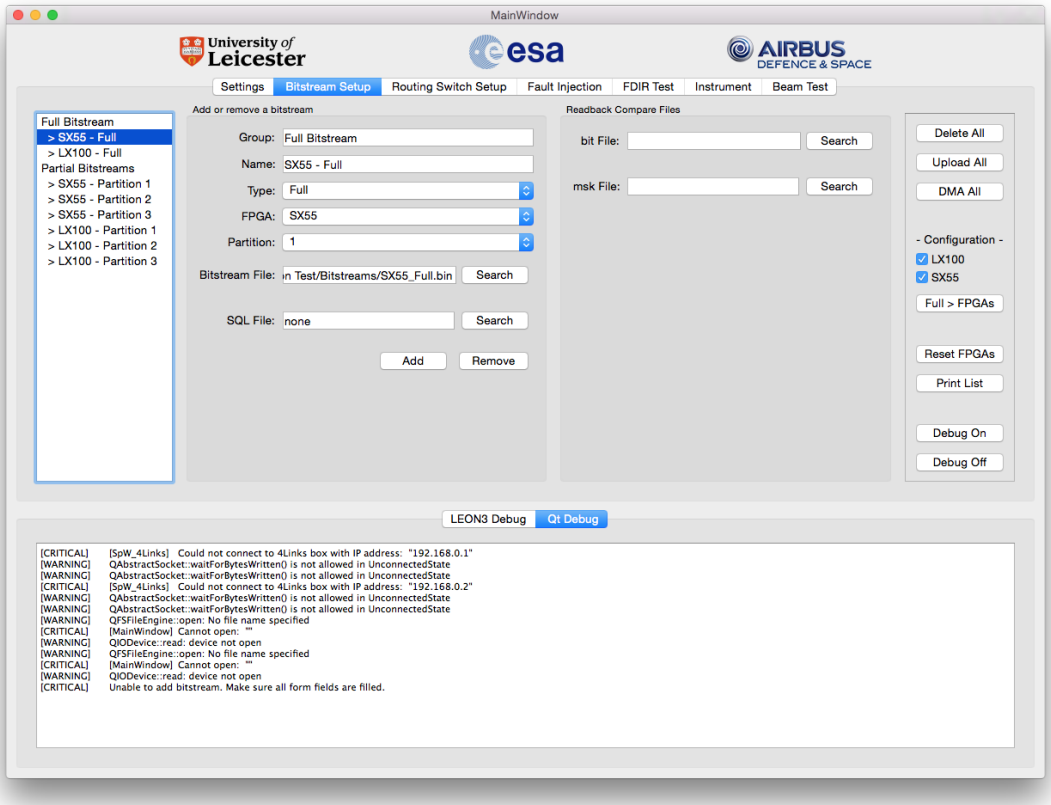

Figure 48: Screenshot of host PC software: bitstream setup

software is printed to a console. In the following, the content of the seven tabs is briefly explained.

TAB 1 - SETTINGS On this tab, the communication interfaces are configured, for instance the IP addresses of the Ethernet to SpaceWire bridges by manufacturer 4Links, which are used to communicate with the embedded systems. The bridges use a simple TCP protocol and an appropriate custom-built driver has been implemented, based on the QTcpSocket class.

TAB 2 - BITSTREAM SETUP On the second tab, the user can enter information about bitstreams into a database, including the type (full vs. partial), the FPGA, the partition, the local file name, and the associated SQLite [148] database file that is used for fault injection results. A screenshot of the tab is depicted in Figure 48. Via the buttons on the right hand side, the following commands can be triggered:

- `Delete All`: Deletes all files from the DRPM file system.

- `Upload All`: Uploads all bitstreams from the database to the DRPM file system.
- `DMA All`: Creates DMA-capable versions of all bitstreams in the DRPM file system.
- `Full > FPGAs`: Executes a full configuration of a specific FPGA. Which FPGA is configured can be selected by ticking checkboxes.
- `Reset FPGAs`: Resets both FPGAs.
- `Print List`: Prints the content of the file system to the console in the bottom half of the tool.
- `Debug On/Off`: Switches the debug output from the DRPM system via serial port on/off.

TAB 3 - ROUTING SWITCH SETUP The controls on the third tab allow the configuration of the FDIR routing switches, including the timeout values and the routing tables. Depending on the test bench setup, the user can transmit the configuration packets to the routing switches directly from the host PC or via the instrument simulator software. A message in the console window informs the user if the transmission of the values was successful.

TAB 4 - FAULT INJECTION This tab comprises all controls needed for the automatic and manual fault injection as well as for the voter verification:

- For the automatic fault injection, a partial bitstream must be chosen. Then, the software analyses the bitstream and creates a list of bitstream byte offsets and corresponding FAR addresses. The offsets are necessary to "cut out" a configuration frame from the partial bitstream that is stored in the file system of the DRPM whereas the FAR address is needed to download the configuration frame to the right memory position within the FPGA's configuration memory. Unfortunately, the relation of the FAR address to the byte offset is not linear and thus knowledge about the bitstream layout is needed. Public information can be found in the reference design file package attached to application note XAPP988 [28]. During bitstream analysis, clock and IOB related configuration frames are skipped. Once all configuration frames are stored in the list, the user can start either a

full or random fault injection campaign. In the first case, faults are injected into the configuration memory bit by bit, starting from a desired bit position. In the latter case, the software picks a random frame from the list and a random bit from within the frame. Beforehand, however, a continuous data stream through the stream processor must be started, which can either be provided by the host PC software or the embedded instrument simulator software. Then, the software executes the fault injection campaign as proposed in Section 4.3.1. However, to speed up the campaign, the latest software version does not classify the configuration bit. Instead, it is only tested if a bit is sensitive or not. This information is stored in a SQLite database and can later be used for voter verification and other tasks.

- For debug purposes, the user can also execute a single fault injection by manually entering a FAR address, byte offset and bit position.

- As briefly described in Section 5.4.1, the FDIR supervisor software includes a voter verification task that can be controlled from this tab too. First, the user must create a binary version of the SQLite database mentioned above, containing only the sensitive bits, and upload it to the file system of the DRPM. Then, the user can choose a specific voter module and the slot that is associated with the stream processor, into which the faults are injected. After starting the campaign, the FDIR supervisor software injects faults into each sensitive bit of the chosen stream processor bitstream and checks if the voter handles the resulting failure correctly. Status information is constantly transmitted to the host PC software and printed to a dedicated console. It includes the detection time, the correction time as well as a list of all faults that did not lead to a failure.

TAB 5 - FDIR TEST This tab mainly comprises controls for the bitstream file mapping and the live demonstration mode. A screenshot is depicted in Figure 49.

- The bitstream file mapping function allows the user to map particular bitstreams to particular voter slots. In addition, it can be chosen which slots are handled by the failure recovery routine. Both is important for the FDIR supervisor software as it is oth-

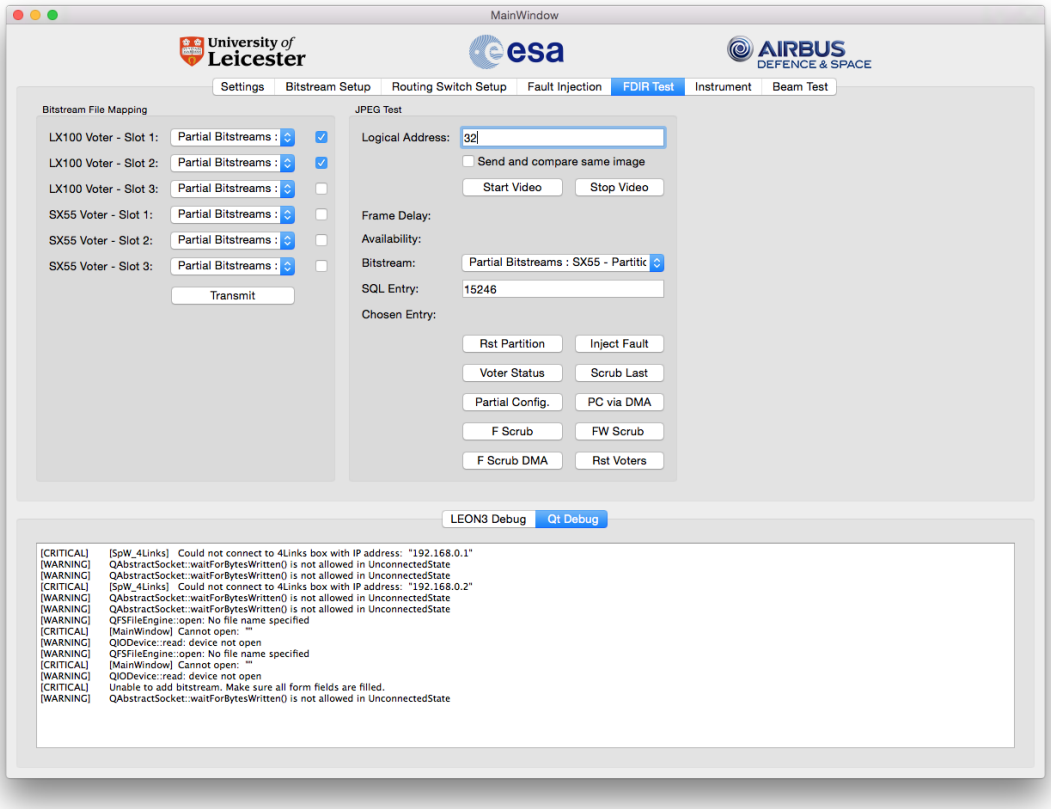

Figure 49: Screenshot of host PC software: FDIR test

erwise not aware of which stream processor is transmitting its output to which voter slot.

- The live demonstration mode allows the user to send a video stream through a stream processor chain. The compressed video is then rendered to screen in a separate window. The raw image data is retrieved from a webcam connected to the host PC using the OpenCV library [149]. The live demonstration mode is especially effective if no redundancy is applied to a stream processor. Then, the user can inject faults into the processor and the effects become visible in the video stream. To do so, the following controls are available:
  - A fault can be injected into a known to be sensitive bit of a particular stream processor bitstream. The sensitive bit is taken from the aforementioned SQLite database file.
  - The last frame in which a fault was injected can be scrubbed.
  - A partial reconfiguration of the stream processor can either be done via CPU or DMA transfer.
  - The partition can be reset.
  - The health statuses of the voter modules can be reset.

TAB 6 - INSTRUMENT The controls on the sixth tab are used to communicate with the instrument simulator software. First, the user can retrieve a raw image from the webcam (or alternatively use an already existing one). Then, this image can be uploaded to the DRPM file system. Next, the file can be transferred to the instrument simulator, which in turn creates a golden JPEG file. If desired, the user can command the instrument simulator software to transfer the golden JPEG file back to the DRPM file system, from which it can be retrieved to the host PC. One button allows the user to start and stop the periodic transmission of raw test images, another button resets the availability counters. The number of transmitted and correctly received images as well as some other status information is constantly received from the instrument simulator software and printed to a dedicated console.

TAB 7 - BEAM TEST This tab gives quick access to the controls needed during the irradiation test campaign described in Section 8:

- One button triggers a script, which sets up the whole test bench automatically, i.e. the bitstreams are uploaded, the FPGAs are configured, the routing switch tables and timeout values are set, the bitstream files are mapped and the raw test image is uploaded to the instrument simulator.
- The user can enter the file path to a SQLite database file, in which the status information from the FDIR supervisor software is collected, including the time stamp, the time to failure value, the time needed for failure recovery as well as the file name of the bitstream that was read back from the Virtex-4 FPGA after failure detection. Once the test is running, the content of this database file is shown in a table, which is updated in real-time.
- The user can enter the file path to another SQLite database file, in which the status information from the instrument simulator software is collected, mainly the time stamp, the number of transmitted images as well as the number of correctly received images. Again, this information is shown in a table during the test.
- The user can enter the path to a directory, in which the readback bitstreams are stored. The file naming is based on the transmission time stamp.
- The user can start and stop the test by pressing corresponding buttons.
- During development, the proton beam was mimicked by fault injection. For this reason, a button allows the user to start or stop periodic fault injection with a given frequency.

5.4.4 *Block RAM Profiling Software*

The Block RAM profiling software is an essential part of the proposed availability analysis method, see Section 4.3.2. It comprises two parts: (i) a script file (*.do file) for the HDL simulator ModelSim, (ii) a profiling tool written in C++ using the Qt framework, of which a screenshot is shown in Figure 50.

SCRIPT FILE First, a post place & route simulation model is compiled and loaded in ModelSim. Then, the script file is loaded, which executes the following steps:

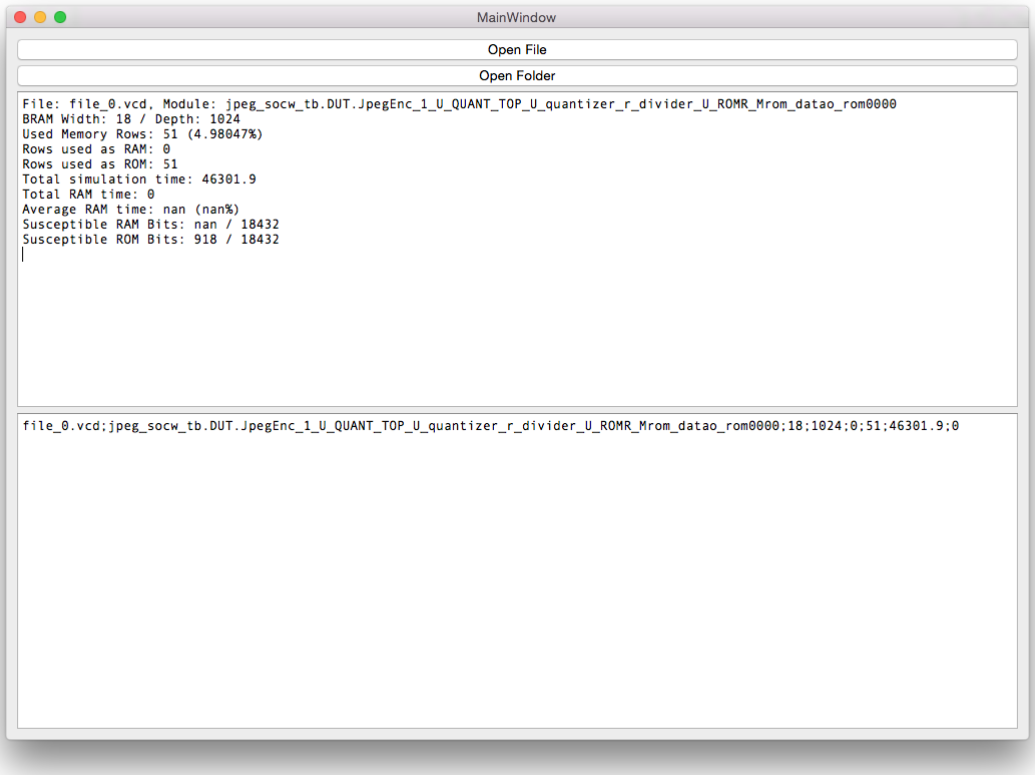

Figure 50: Screenshot of Block RAM profiling software

- The simulation model is searched for all Block RAM instances.
- A waveform file (*.vcd file) and an information file (*.nfo) is created for each Block RAM instance.
- The following signals are added to each waveform file:
  - `ADDRA`: Memory address port A.
  - `ENA`: Block RAM enable port A.
  - `WEA`: Write enable port A.
  - `ADDRB`: Memory address port B.
  - `ENB`: Block RAM enable port B.
  - `WEB`: Write enable port B.
- The following information is collected for each Block RAM instance and stored in the correspondent information file:
  - `WRITE_MODE_A`: Write mode port A (write first vs. read first mode).
  - `WRITE_WIDTH_A`: Write width of port A.
  - `READ_WIDTH_A`: Read width of port A.
  - `WRITE_MODE_B`: Write mode port B (write first vs. read first mode).
  - `WRITE_WIDTH_B`: Write width of port B.
  - `READ_WIDTH_B`: Read width of port B.
  - `DIA_CONNECTED`: Indicates if port A data input is connected.
  - `DIB_CONNECTED`: Indicates if port B data input is connected.
  - `DOA_CONNECTED`: Indicates if port A data output is connected.
  - `DOB_CONNECTED`: Indicates if port B data output is connected.

PROFILING TOOL The algorithm of the Block RAM profiling tool comprises four C++ classes as can be seen in the Unified Modeling Language (UML) diagram in Figure 51. Core of the tool is an object of class `VCDParser`. The main steps of the algorithm are as follows:

- The waveform file is opened and analysed line by line (the file format is defined in IEEE 1364-2005 [150]).
- Each file describes a module, which in this case is a Block RAM instance. For this module, an object of class `VCDModule` is created.

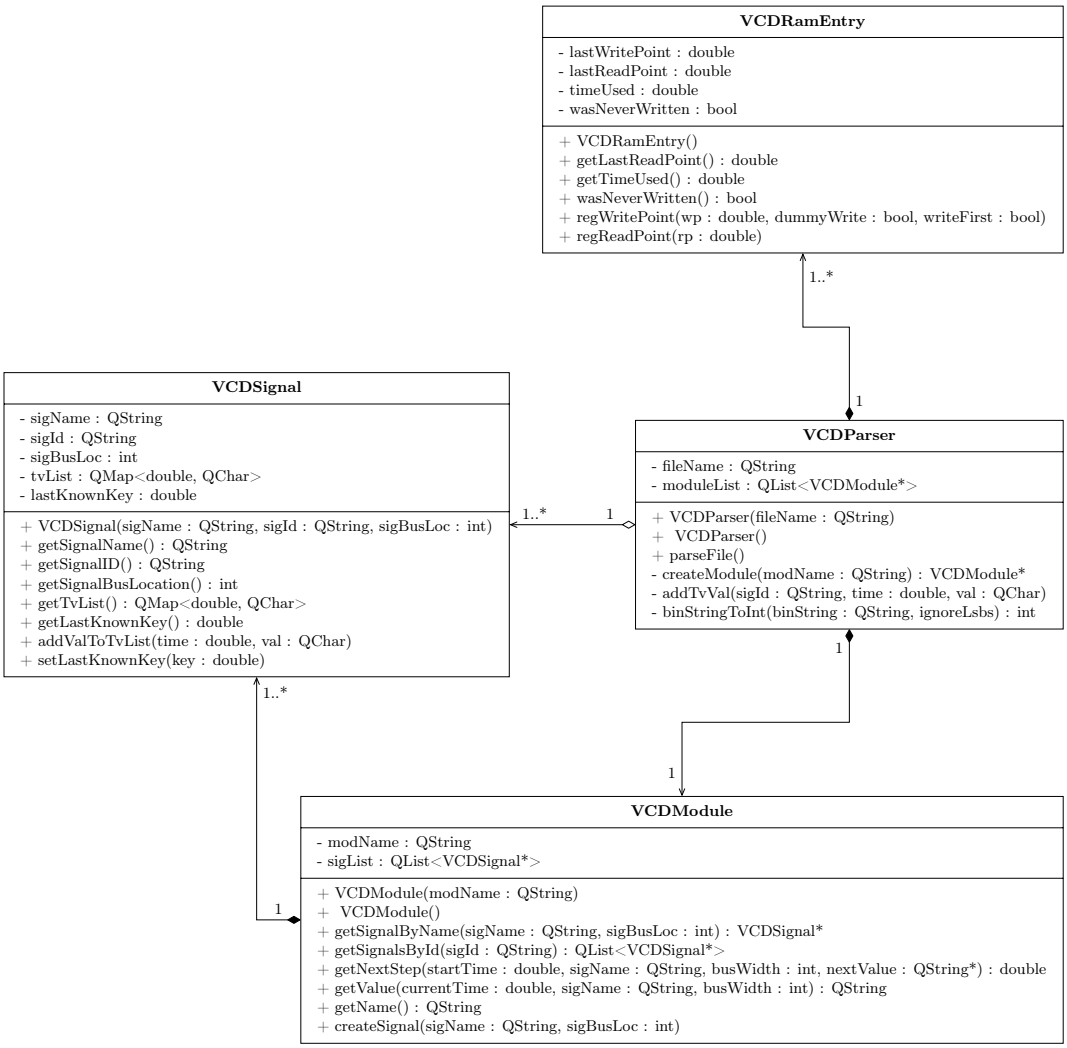

Figure 51: UML diagram of Block RAM profiling tool

- In the header part of the waveform file, the signals are defined (`ADDRA`, `ADDRB` etc.). Buses are split into single wires and each wire carries a unique identifier that is used in the body part of the file. For each wire, an object of class `VCDSignal` is created.
- In the body part, all signal changes over time are listed. While parsing the body part of the file, value changes for specific signals are stored in a list by calling the `addValToTvList(..)` method of the corresponding `VCDSignal` objects.
- Once the waveform file is parsed, the algorithm analyses the information file. For each memory cell of the Block RAM, an object of class `VCDRamEntry` is created.
- Now, the actual Block RAM profiling begins. The tool steps through the waveform data from the beginning to the end of the simulation run. Depending on the configuration of the Block RAM, signal changes can occur on the write enable signals of port A and B and on the memory address buses of port A and B. By calling the `getNextStep(..)` method of the `VCDModule` object for each signal/bus mentioned above, the next minimal time step, at which a signal change occurs, can be determined.
- If a write enable for port A or B is active at the current time step, the corresponding memory address is determined and the memory write is registered for this memory cell by calling the `regWritePoint(..)` method of the corresponding `VCDRamEntry` object.
- Similarly, read enables for the ports are registered by calling the `regReadPoint(..)` method of the corresponding `VCDRamEntry` object.
- Every time a new write point is registered for a specific memory cell, the time span between the last write point and the last read point is added to an accumulation register. The accumulated time stored in this register represents the time, in which the memory cell is susceptible, see also Section 4.3.2.
- Once the complete simulation run is analysed, the above mentioned accumulated times of each memory cell are averaged over all memory cells of the Block RAM. Dividing this averaged time span by the total simulation time leads to the correction factor T_S described in Section 4.3.2.

```
File: file_82.vcd
Module: jpeg_socw_tb.DUT.JpegEnc_1_U_QUANT_TOP_U_RAMZ_Mram_mem
BRAM Width: 18 / Depth: 1024
Used Memory Rows: 128 (12.5%)
Rows used as RAM: 128
Rows used as ROM: 0
Total simulation time: 46303.9
Total RAM time: 417315
Average RAM time: 3260.27 (7.04103%)
Susceptible RAM Bits: 162.225 / 18432
Susceptible ROM Bits: 0 / 18432
```

Figure 52: Example output of Block RAM profiling tool

- Then, the number of sensitive RAM bits can be determined by multiplying the correction factor by the number of used RAM bits.

An example output of the Block RAM profiling tool for one waveform file is shown in Figure 52.

First, the tool lists the file and module name, the Block RAM width and the Block RAM depth. Then, it lists the number of memory rows used as RAMs and ROMs. Next, it lists the total simulation time and the average time span in which a memory cell is susceptible (the percentage value in brackets is the correction factor T_S). Finally, the absolute number of susceptible RAM and ROM bits is given.

5.5 CONCLUSIONS

A complex proof of concept system based on a mix of different technologies was implemented, which comprises components that are very similar to the ones found in spaceborne hard- and software, e.g.:

- Virtex-4 SRAM-based FPGAs are used, which are available in space-qualified versions.
- The ProASIC3e device that implements the FDIR supervisor SoC design could easily be replaced by a radiation-hardened anti-fuse FPGA.
- All network communication is done via SpaceWire, a network protocol found in many modern space engineering projects.
- The JPEG image compression stream processor is a good example for typical satellite payload data processing applications.

- The embedded FDIR supervisor software is written for RTEMS, a popular real time operating system for on-board computers.

This realistic demonstration system implements for the first time the Distributed Failure Detection method in a multi-FPGA system. As a result of that, it becomes possible to validate both the Distributed Failure Detection and the Availability Analysis method, proposed in Chapter 3 and 4, respectively. In the following chapters, the system is used for several important purposes: first to obtain power and performance measurements, then for the availability analysis of the implemented stream processor (using fault injection experiments and the Block RAM profiling method) and finally as a test bench for an accelerated proton irradiation test campaign.

6

CHAPTER 6: EVALUATION OF FDIR MECHANISMS

6.1 INTRODUCTION

A multitude of failure detection, masking and recovery techniques for SRAM-based FPGAs have been proposed in the literature. Although power, area or performance overhead figures are sometimes given, they are usually not suitable for comparing the techniques fairly since a variety of incompatible measurement setups and terminologies are used by the different authors.

In Section 2.9 the claim was made that modular redundancy is the best approach for the FDIR methodology proposed in the course of this PhD work. Now, with the proof of concept implementation described in the previous Chapter 5, it is feasible to prove if this is also correct in terms of power, area and performance overhead. Hence, this chapter aims at measuring these overheads for several popular mitigation techniques applied to the same benchmark circuit.

The chapter is structured as follows. In Section 6.2, an overview of the power measurement setup is given. The different failure detection, masking and recovery techniques that are examined in this chapter are explained too. In Section 6.3, the measurement results are given. Section 6.4 discusses the results before the chapter is finally concluded in Section 6.5.

6.2 EXPERIMENTAL METHODOLOGY

6.2.1 *System Overview and Power Measurement Setup*

The essential parts of the DRPM system needed for the following measurements are outlined in Figure 53. For all measurements, the second FPGA device (LX100) remained unconfigured. As mentioned earlier, the SoC on the ProASIC3e device implements a LEON3 microprocessor that is connected to two memories via AMBA bus. The 2 GBit SD-RAM memory stores the bitstreams for the Virtex-4 FPGAs, the smaller MRAM is utilised as main memory by the microprocessor. A NoC routing switch allows the system to transfer data to the

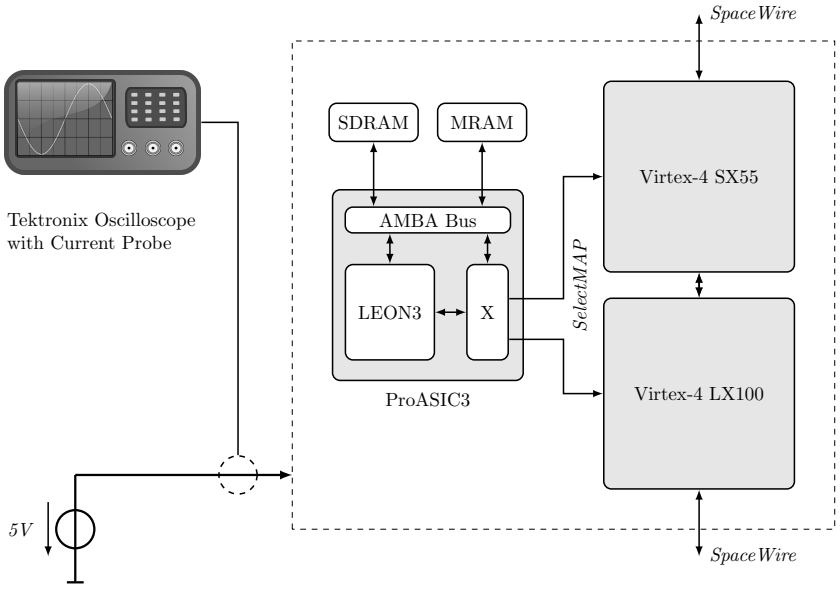

Figure 53: Simplified overview of power measurement setup

SelectMAP configuration interfaces of the two Virtex-4 FPGAs. This can either be done via DMA transfer or via CPU. In the first case, the SD-RAM controller is a bus master, which can transfer data from memory directly to one of the SelectMAP interfaces. In the latter case, the microprocessor retrieves data from SD-RAM, storing it in MRAM temporarily, before it is finally transferred to one of the SelectMAP interfaces. The AMBA bus and the SelectMAP interfaces run at 40 MHz whereas the LEON3 microprocessor runs at 20 MHz.

The power consumption is measured by a Tektronix TDS5054B oscilloscope using a high precision current probe TCP312. The 5 V power line is tapped with the current probe, i.e. the power consumption of all components on the PCB is measured. This type of setup was chosen on purpose because the real power consumption of the mitigation techniques is of interest, which also includes some CPU load and external memory accesses.

6.2.2 *Failure Masking and Detection*

Two basic approaches for failure masking and detection are described in literature, both based on spatial redundancy. The first approach applies TMR to the netlist of a circuit whereas the second approach implements a modular redundancy technique, i.e. a whole hardware block is multiplied. For the measurements, both approaches have

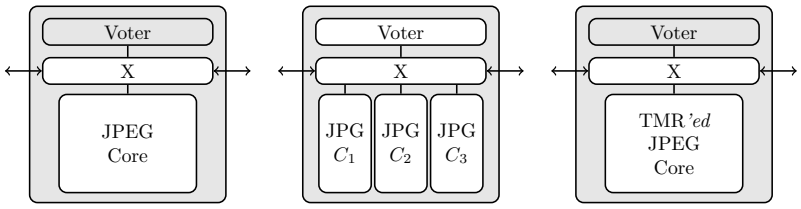

Figure 54: Failure masking and detection designs. From left to right: (a) no redundancy, (b) modular TMR, (c) netlist TMR

been implemented, see also Figure 54. The design implemented on the SX55 device comprises a NoC routing switch X that interconnects the FPGA with external devices. The JPEG image compression stream processor was used as benchmark circuit, which is connected to the NoC switch too. In design (a), no redundancy is applied to the stream processor. In design (b), the whole stream processor is triplicated and each instance C_n is connected to the routing switch. In design (c), TMR is applied to the netlist of the JPEG stream processor, i.e. the voting is done within the JPEG image compression block. First, the JPEG core was synthesised as a stand-alone module. The resulting netlist was then used as input to the TMRTool 13.2 by Xilinx [151], which applied TMR to the netlist using the default settings. The JPEG core has a fair amount of state-dependent logic (like state machines, counter etc.) and therefore 833 inserted one-bit voters could be counted in the modified netlist. Finally, the netlist was integrated into the full FPGA design. To allow a fair comparison, all three designs comprise the same components except that only in design (b) the voter module within the FDIR routing switch is clocked and therefore active.

6.2.3 *Failure Recovery*

Regarding SRAM-based FPGAs, failure recovery is mainly done by refreshing the configuration memory.

In case of modular TMR, partial reconfiguration is a popular recovery scheme. If a partial bitstream is downloaded to the FPGA device, it does not only include the configuration memory content but also the user memory initialisation values, i.e. the start-up content of the flip-flops and the Block RAMs.

In case of netlist TMR, a partial reconfiguration of a faulty partition is not possible since no physical separation between the partitions exists. Thus, a memory refreshing technique called scrubbing is often implemented. Although quite similar to a regular partial reconfigura-

Table 10: Area: Failure masking

Design	Slices	RAMBs	DSP48s
(a) No Redundancy	9,953 (40%)	90 (28%)	10 (1%)
(b) Modular TMR	21,852 (88%)	260 (81%)	30 (5%)
(c) Netlist TMR	20,909 (85%)	258 (80%)	30 (5%)

tion, the start-up content of user memories is not written to the FPGA and as a result, scrubbing can be executed during normal circuit operation. For the measurements presented here, two scrubber types have been implemented according to a Xilinx application note [28]. The first one is a device-based blind scrubber, which downloads a full bitstream to the device. The second one is a frame-based scrubber, which downloads the content of the full bitstream frame by frame.

Except of the frame-based scrubber, all aforementioned recovery techniques are implemented as pure hardware solutions (via DMA transfer) and as software solutions (via CPU). In the first case, data transfers to the SelectMAP interface are rather fast since the 8-bit wide interface runs at 40 MHz. The software-based scrubbers, however, are slow due to the lower CPU frequency of 20 MHz and the increased number of required memory accesses.

Due to technical limitations, the frame-based scrubber is only implemented in software. It reads a block of data, representing a configuration frame, from the full bitstream stored in SD-RAM. Then, it appends the configuration frame header and an empty dummy frame that is required to flush the pipeline of the configuration engine. The FAR address of each frame is calculated in software. Due to the heterogeneity of Virtex-4 devices, the calculation is rather complex and time consuming. As an alternative, the prepared configuration frames could be pre-stored in SD-RAM or the FAR addresses of all configuration frames could be stored in a look-up table. However, both approaches would necessitate increased memory resources, which are usually rare in space electronics systems.

Table 11: Performance: Failure masking

Design	Min. Period	Max. Frequency
(a) No Redundancy	9.04 ns	110.66 MHz
(b) Modular TMR	9.52 ns	105.02 MHz
(c) Netlist TMR	10.10 ns	98.97 MHz

6.3 RESULTS

6.3.1 *Failure Masking and Detection*

6.3.1.1 *Area Overhead*

Table 10 lists the resources that are needed for the implementation of the three designs. Although many one-bit voters are added to the netlist of the design, the netlist TMR approach (c) needs slightly less slices than the modular TMR approach. In addition, two Block RAMs are saved. It can be assumed that the voters, which are implemented in LUTs, can easily be mapped into slices with unused LUTs. In contrast, the modular TMR approach constrains the areas of the partitions physically. Thus, the toolchain is unable to optimise components across the partition boundaries, which could also explain why two Block RAMs are saved when applying the netlist TMR approach.

6.3.1.2 *Performance Overhead*

Table 11 lists the performance overhead of each design. All figures given are post-place & route and were gathered by incrementing the timing constraints step by step until the place & route tool was unable to meet the constraints. Unsurprisingly, the design to which no redundancy was applied performs the best and can reach a clock frequency of 110.7 MHz. The modular TMR approach reaches a lower clock frequency of 105 MHz, again most likely due to the physical area constraints, which limit the toolchain in its place and route efforts. Another drop of performance is observed for the netlist TMR approach, which can only reach less than 99 MHz. Since this design was not able to meet the planned target frequency of 100 MHz, a DCM was added to all three designs that decreases the clock frequency to the next possible step. Therefore, all following power measurements were conducted at 98.875 MHz.

Table 12: Power: Failure masking (no data processing)

Design	Current [A]	Power [W]	Rel. Power [W]
(o) Not configured	0.94	4.70	0.00
(a) No Redundancy	1.14	5.71	+1.01
(b) Modular TMR	1.31	6.57	+1.87
(c) Netlist TMR	1.36	6.78	+2.08

Table 13: Power: Failure masking (active data processing)

Design	Current [A]	Power [W]	Rel. Power [W]
(o) Not configured	0.94	4.70	0.00
(a) No Redundancy	1.19	5.94	+1.24
(b) Modular TMR	1.34	6.69	+1.99
(c) Netlist TMR	1.36	6.82	+2.12

6.3.1.3 *Power Overhead*

Table 12 lists the power overheads of all designs that are clocked but do not process any data. In contrast, Table 13 lists the overhead during active data processing. First, the static power of the PCB was measured by keeping both SRAM-based FPGAs unconfigured. Although the LEON3 microprocessor was running, it did not execute any demanding tasks or memory accesses. The average power measured in this setup was 4.7 W. Then, the SX55 device was configured with design (a), which led to a power consumption of 5.7 W, an increase of ca. 1 W compared to the unconfigured design. Afterwards, the FPGA was configured with the modular TMR design (b), resulting in a power consumption of 6.6 W, an increase of nearly 1.9 W. The netlist TMR design (c) could even exceed this result by another 0.2 W.

Naturally, the power consumption increases during active data processing, as can be seen in Table 13. However, the increase seems to depend on the design. Although the power overhead of all TMR approaches contains a rather large static part, this is particularly true for the netlist TMR design.

One benefit of modular TMR is the fact that redundant instances can be switched off on-demand if power must be saved. To measure how much power can be saved, blank bitstreams were created for each partition that allow the removal of stream processors from the

Table 14: Power: Modular TMR (no data processing)

Design	Current [A]	Power [W]	Rel. Power [W]
(0) Not configured	0.94	4.70	0.00
(1) All blanked	1.05	5.26	+0.56
(2) C_1	1.14	5.71	+1.01
(3) C_2	1.14	5.70	+1.00
(4) C_3	1.14	5.71	+1.01
(5) $C_1 + C_2$	1.23	6.13	+1.43
(6) $C_1 + C_3$	1.23	6.16	+1.46
(7) $C_2 + C_3$	1.23	6.14	+1.44
(8) $C_1 + C_2 + C_3$	1.31	6.57	+1.87

Table 15: Power: Modular TMR (active data processing)

Design	Current [A]	Power [W]	Rel. Power [W]
(0) Not configured	0.94	4.70	0.00
(1) All blanked	1.05	5.27	+0.57
(2) C_1	1.15	5.76	+1.06
(3) C_2	1.15	5.75	+1.05
(4) C_3	1.15	5.76	+1.06
(5) $C_1 + C_2$	1.25	6.23	+1.53
(6) $C_1 + C_3$	1.25	6.24	+1.54
(7) $C_2 + C_3$	1.25	6.23	+1.53
(8) $C_1 + C_2 + C_3$	1.34	6.69	+1.99

design. The results are listed in Table 14 and 15, again for phases without data processing and phases with active data processing.

If all partitions are blanked, the average power consumption is 5.3 W, i.e. the static area of the FPGA design consumes approximately 0.6 W. If one of the cores is installed, a further increase of 0.45 W can be observed. Installing all three JPEG cores leads to a more or less linear increase with an overall power consumption of 6.6 W. During active data processing, the power consumption increases linearly too.

6.3.2 *Failure Recovery*

Table 16 lists the times needed for each failure recovery approach. A partial reconfiguration of one of the JPEG stream processors can be

Table 16: Performance: Failure recovery

Design	Time (CPU) [s]	Time (DMA) [ms]
(a1) Partial Reconfiguration C_1	0.53	27.5
(a2) Partial Reconfiguration C_2	0.53	27.5
(a3) Partial Reconfiguration C_3	0.55	27.5
(b) Device-based Scrubbing	1.44	129
(c) Frame-based Scrubbing	16.93	

done in less than 30 ms via DMA transfer but takes more than half a second if executed by the microprocessor. The long execution time is due to the fact that the CPU must first load a block of configuration data from SD-RAM to MRAM before it is finally downloaded to the SRAM-based FPGA. As the size of one data block is only 4 kB and the clock frequency of the CPU is low, low performance must be expected. Similarly, device-based scrubbing can be executed in 129 ms when using DMA transfer whereas the same transfer takes 1.44 s via the microprocessor. Device-based scrubbing can be quicker than partial reconfiguration though because the partial bitstream also contains the start-up content of the user memory elements. Due to the chosen implementation, the frame-based scrubber is extremely slow. Aside from the increased processing time needed by the CPU, each frame must be flushed with an empty frame. Together with the required frame header, the data volume that must be transferred to the FPGA is effectively doubled. In this implementation, the FAR address for each frame is calculated in software which explains the extremely low performance of this solution.

The given results should be understood in a qualitative manner due to their strong dependency on the chosen implementation. It is worth to point out, however, that the time needed to execute a failure recovery action can be much longer than possibly expected, especially since the performance of space qualified components is rather low. It is also worth to point out that software-based solutions can bear a high performance overhead.

Table 17 lists the power overhead of each failure recovery approach that was measured while the system was running in a modular TMR configuration with three active JPEG cores. The base power needed during normal system operation was 6.69 W. During partial reconfiguration and device-based scrubbing via CPU an increase of ca. 0.5

Table 17: Power: Failure recovery

Design	Power (CPU) [W]	Power (DMA) [W]
(o) System running	6.69	
(a) Partial Reconfiguration	7.18 (+0.49)	7.02 (+0.33)
(b) Device-based Scrubbing	7.21 (+0.52)	6.99 (+0.30)
(c) Frame-based Scrubbing	7.31 (+0.62)	

W was observed. The increased power overhead includes operations executed by the CPU, read accesses to the SD-RAM, write and read accesses to the MRAM as well as write accesses to the configuration memory of the SRAM-based FPGA. The frame-based scrubber consumes even 0.6 W, which is most likely due to the additional processing overhead required for the calculation of the FAR addresses. In contrast, the same failure recovery approaches executed via DMA transfer led to a power overhead increase of only 0.3 W since they only include read accesses to the SD-RAM and write accesses to the configuration memory.

6.4 DISCUSSION

6.4.1 *Failure Masking and Detection*

The simple rule that a triplicated circuit also consumes two times more power and resources can be confirmed. There are, however, differences between the TMR implementations. It seems as if the netlist TMR approach requires slightly less resources than the modular TMR approach, which is most likely due to the area constraints applied in the latter approach, which limit the place and route capabilities of the toolchain. On the other hand, the modular TMR approach offers better performance. This is due to the fact that many routes must be added for the inserted one-bit voters in the netlist TMR approach, which potentially increases the critical path length. The modular TMR approach also consumes less power than the netlist TMR approach.

One advantage of the modular TMR approach is its capability to make a system adaptable to reliability and power consumption constraints, respectively, because redundant instances can be added or removed on demand. In case of the JPEG stream processor, approximately 0.5 W can be saved for each removed redundant instance.

With its better performance and lower power overhead, together with the possibility to further save power by temporally switching off redundant instances, the modular TMR approach seems to be well suited for payload data processing applications, in which high performance might be a more important factor than system availability. Systems with high availability requirements, however, have to buy the increased reliability at the cost of maximum power consumption and minimum performance.

6.4.2 *Failure Recovery*

Reconfiguring SRAM-based FPGAs can be quite expensive in terms of power consumption. In the DMA implementation an average power consumption of 0.3 W was observed whereas the CPU implementation reached 0.5 W, a considerable figure taking into account that the JPEG stream processor consumes the same amount of power. In reality, the FPGA is usually scrubbed with low frequency, ranging from one scrub cycle per every few seconds to minutes or even hours. Then, the average power consumption becomes negligible. However, some publications suggest a high "non-stop" scrubbing frequency. While this might increase the system availability, one should take the power consumption into account when considering such an approach.

If a stream processor in a modular TMR configuration is only repaired once a failure has been detected (as it is the case in the proposed FDIR methodology), the power consumption is minimised since no unnecessary memory accesses are executed. In addition, repairing one single stream processor is quicker than executing a device-based scrubbing cycle.

Regardless of the chosen failure recovery approach, a hardware-based implementation will usually perform better than a software-based implementation. On the one hand, the power required during reconfiguration is most likely decreased since less memory accesses are needed. On the other hand, the failure recovery action can be executed much quicker, which leads to a lower average power consumption over time.

6.5 CONCLUSIONS

Several FDIR techniques were evaluated in this chapter. The evaluation approach is novel, as failure detection and failure recovery

approaches are compared fairly in terms of power, area and performance overhead for the first time.

Two popular TMR approaches, often described in literature, were compared in terms of power, area and performance overhead. For the used benchmark circuit, modular TMR performs better than netlist TMR in terms of achievable clock frequency and power consumption but requires slightly more logic resources. Hence, this approach seems to be well suited for high performance payload data processing applications as targeted by the proposed FDIR methodology.

It was found that the reconfiguration process can be expensive in terms of power consumption if necessary external memories are also taken into account. This result further supports the idea of using a real FDIR approach, in which faulty stream processors are only repaired once a failure has been detected. If this is not the case, the scrubbing rate should be adapted to the mission's constraints in terms of availability and power overhead. As it will be shown in Section 7.5.1 of the following chapter, a scrubbing interval of approximately 8 minutes can already be sufficient even for a design without a redundancy scheme. If the design is protected by a netlist TMR approach, the interval could be increased substantially to several hours or even days.

In general, a hardware-based implementation of the reconfiguration controller performs much better than a software-based implementation.

7

CHAPTER 7: AVAILABILITY ANALYSIS CASE STUDY

7.1 INTRODUCTION

This chapter aims at demonstrating how the Availability Analysis method, proposed in Chapter 4, is applied in practice. The necessary parameters are determined for the proof of concept implementation, introduced in Chapter 5, step by step. For simplicity, only the two big contributors to the steady-state availability are taken into account: The configuration memory and the Block RAMs. Furthermore, only dynamic partial reconfiguration is used as failure recovery strategy. For illustrative purposes, the steady-state availability figures for two European satellite missions are calculated for each redundancy configuration. By doing so, the capabilities of each FDIR approach can easily be compared.

The chapter is structured as follows. Section 7.2 gives an overview of early fault injection experiments that were used to validate the FDIR components of the proof of concept system. In Section 7.3, the SEU rates for the configuration memory and the Block RAMs are computed for the two example satellite missions. Then, in Section 7.4, the number of sensitive configuration memory and Block RAMs cells within the JPEG stream processor is determined, which allows the calculation of the MTBF figures that must be expected during the two example satellite missions. Next, the steady-state availability of the stream processor in three different redundancy configurations and for both example satellite missions is determined in Section 7.5. Section 7.6 discusses the results before the positive influence of the Block RAM profiling tool on the prediction precision is illustrated in Section 7.7. Finally, Section 7.8 concludes the chapter.

7.2 PRELIMINARY FAULT INJECTION EXPERIMENTS

In the early development phase of the proof of concept system, many fault injection experiments were conducted to gain a better understanding of the extend of SEU-induced data corruption. Some exam-

Figure 55: Corrupted JPEG images due to fault injections

ples of corrupted JPEG images are shown in Figure 55. As can be seen, only a single bit flip in the configuration memory can have dramatic effects on the algorithm.

Later on, an automatic fault injection system was implemented that was at that time slightly different to the one proposed in Section 4.3.1. Instead of using the voter module as failure detector, failures in the network stream were detected by the host PC software by comparing the returned JPEG images to a golden copy. The results of the fault injection campaign were stored in a SQLite database, including the classification of the sensitive bits.

Once the voter module was implemented and integrated into the FDIR routing switch, this SQLite database turned out to be very useful for the verification of the voter module. As described in Section 5.4.1, the FDIR supervisor software includes a task for the voter verification. Using the aforementioned SQLite database as input, this task was able to verify that the voter module detects the same bits as being sensitive as the early fault injection system did before. Interestingly, a very small number of sensitive bits did not lead to failures during this verification campaign. It turned out that in extremely rare cases the manifestation of failures due to SEUs can be data-dependent. For instance, if a darker image is processed the upset might not lead to a failure but once a lighter image is processed the same upset might manifest itself as a failure. In the following, this data-dependency is not taken into account because of its rareness. However, the situation might be different for other kinds of algorithms and should be thus kept in mind.

7.3 RADIATION ENVIRONMENTS

To estimate the availability of the JPEG stream processor in a specific radiation environment, the heavy ion and/or proton fluxes of this environment must first be known. As proposed in Section 4.4.1, these fluxes are calculated according to standard ECSS-E-ST-10-04C.

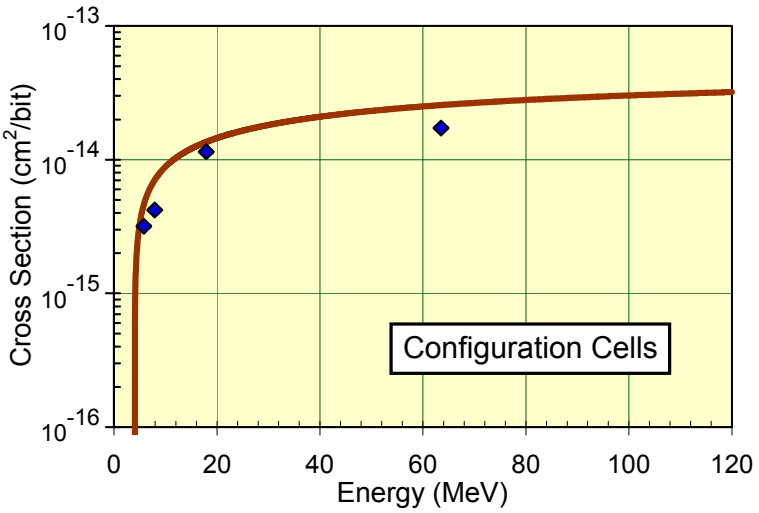

Figure 56: Cross-section for configuration memory cells in XQR4VSX55 devices as provided by NASA/Xilinx [3]

Table 18: SEU rates per bit-day for ESA missions

Mission	Config. Memory	Block RAM
Sentinel-3	7.3E-07	2.3E-06
Sentinel-3 (SPE)	2.9E-04	1.1E-03
Galileo	2.6E-07	5.8E-07
Galileo (SPE)	9.4E-04	3.7E-03

Two satellite missions of the European Space Agency are considered for the case study:

- Sentinel-3, LEO, 814.5 km altitude, 98.65° inclination.
- Galileo, Medium Earth Orbit (MEO), 23,222 km altitude, 56° inclination.

For the two ESA missions, the SEU rates per bit-day are calculated using the OMERE tool, which greatly simplifies the SEU rate prediction according to the aforementioned ECSS standard. OMERE was developed by the French company TRAD with support from the French space agency Centre National d'Études Spatiales (CNES) [152]. Aside from the mission parameters, the cross-sections for the memory elements must be fed into the tool. This can be done by entering the Weibull parameters provided by NASA and Xilinx [3]. The corresponding cross-section curves for proton fluxes are shown in Figure 56 and 57. The computed SEU rates are listed in Table 18: Under normal conditions, the SEU rates for Sentinel-3 are higher than the rates

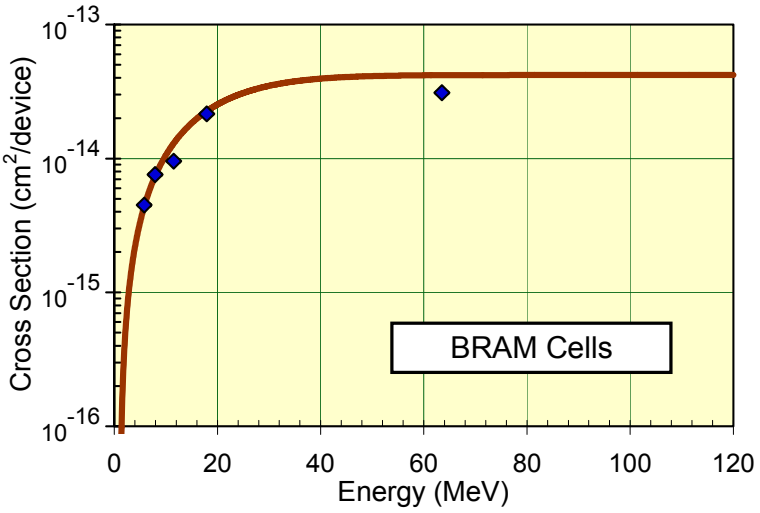

Figure 57: Cross-section for Block RAM cells in XQR4VSX55 devices as provided by NASA/Xilinx [3]

for Galileo due to the influence of the inner radiation belt. The opposite is true for Solar Particle Events (SPEs) as the lower orbit offers a natural protection due to the geomagnetic shielding of the Earth.

7.4 QUANTIFICATION OF SENSITIVE MEMORY ELEMENTS

7.4.1 *Configuration Memory*

To quantify the number of sensitive configuration memory bits, a simplified version of the fault injection system proposed in Section 4.3.1 is used. Since the failure recovery strategy is solely based on partial reconfiguration, a further classification of the sensitive bits (into $F_{C/S}$, $F_{C/SR}$, $F_{C/Re}$) is unnecessary and therefore only the total number of sensitive bits F_C is of interest.

In the same section it was claimed that random fault injection is sufficient for the estimation of the total number of sensitive bits within a bitstream. Using the formula for Wald confidence intervals (see Equation 2), the number of required fault injections can be estimated: Suppose, a typical 95% confidence interval ($z = 1.96$) with an interval width of $\pm 0.2\%$ ($p = \hat{p} \pm 0.002$) is desired while 15% of all configuration bits are guessed to be sensitive ($\hat{p} = 0.15$). Then, n becomes 122,451.

Based on this estimation, a campaign with 150,000 fault injections is chosen, which can be conducted within a couple of hours. In contrast,

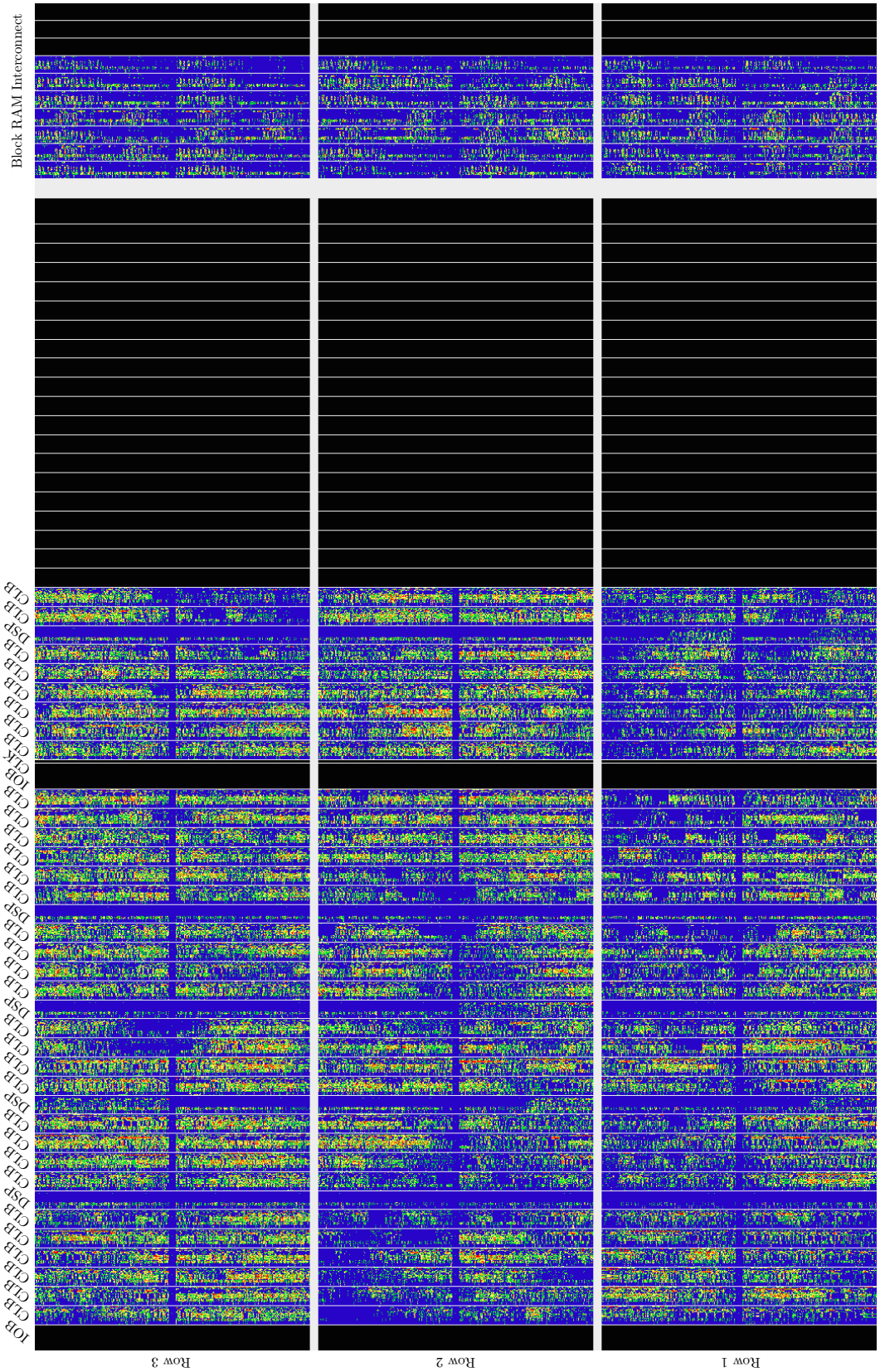

Figure 58: Heat map of fault injection results

Table 19: Random fault injection results

# Tested	# Sensitive	95% confidence interval	
		Min	Max
1,000	134 (13.40%)	11.35%	15.67%
5,000	701 (14.02%)	13.07%	15.01%
10,000	1,382 (13.82%)	13.15%	14.51%
50,000	6,922 (13.84%)	13.54%	14.15%
100,000	13,899 (13.90%)	13.69%	14.11%
150,000	20,870 (13.91%)	13.74%	14.09%

a full fault injection campaign that injects faults into all 3,735,264 bits of the stream processor bitstream would take several weeks.

The results of the random fault injection campaign are given in Table 19. As can be seen, the estimation of the percentage of sensitive bits becomes better with an increased number of fault injections, i.e. the confidence interval width becomes narrower. Using the worst case assumption, the total number of sensitive bits is:

$$F_C = 0.1409 \cdot 3735264\text{bits} = 526299\text{bits} \approx 526300\text{bits} \quad (6)$$

To prove that this estimation is realistic, a full fault injection campaign was conducted as well, which revealed that 523,543 bits (14.02%) are sensitive. Since this result is within all confidence intervals found in Table 19, it is proven that random fault injection can provide accurate results. For illustrative purposes, the fault injection results are visualised in a heat map, see Figure 58. The tested stream processor is hosted on a reconfigurable partition in the top left corner of the Virtex-4 FPGA, spanning vertically row 1 to row 3. The detail shown in the heat map corresponds to the white box labeled "prmodule_1" in Figure 43. Each pixel in the heat map represents 4 bits of a configuration frame. The colour code of the pixels is as follows: (i) blue = tested, (ii) green = 1 bit sensitive, (iii) yellow = 2 bits sensitive, (iv) orange = 3 bits sensitive, (v) red = 4 bits sensitive.

To calculate the MTBF figures for the different radiation environments, the SEU rates from Table 18 must first be multiplied by the total number of sensitive configuration memory bits and then the reciprocal value must be taken from this result. The resulting MTBF figures can be found in Table 20 and express how long it takes in av-

Table 20: MTBF values for configuration memory

Mission	MTBF
Sentinel-3	2.6 days
Sentinel-3 (SPE)	9.4 minutes
Galileo	7.3 days
Galileo (SPE)	2.9 minutes

erage until a failure occurs at the output of the stream processor that was triggered by an upset in the configuration memory.

7.4.2 *Block RAM*

The number of sensitive Block RAM bits is determined using the Block RAM profiling tool. For the theoretical background see Section 4.3.2 and for the practical implementation see Section 5.4.4.

A post place & route simulation model of the JPEG stream processor is simulated in ModelSim. The HDL testbench simulates the processing of one full image, resembling the scenario found in the proof of concept implementation: The data enters the stream processor via SpaceWire with a data rate of 160 Mbit/s and the raw source images are sent periodically every 100 ms. It is especially important to model the frame rate accurately because the processing of one image takes only around 50 ms, i.e. half of the time the Block RAMs of the stream processor are idle and are thus not susceptible to SEUs.

In total, the stream processor comprises 83 Block RAMs. In Virtex-4 devices a Block RAM block comprises 18,432 bits. Therefore, without any knowledge about the usage of the Block RAMs, one could assume a total of 1,529,856 Block RAM bits.

From this large number of total Block RAM bits, the profiling tool revealed that 524,964 bits (34.3%) are used as RAM bits and 27,351 bits (1.8%) as ROM bits.

Now, taking the correction factor τ_s into account, the Block RAM profiling tool predicts that only 67,858 RAM bits (4.4%) must actually be counted as being susceptible. Thus, together with the ROM bits, a total of 95,209 sensitive Block RAM bits is estimated.

Similarly to the configuration memory, the MTBF figures for the different radiation environments are calculated by first multiplying the SEU rates from Table 18 by the total number of sensitive Block RAM bits and then taking the reciprocal value from this result. The

Table 21: MTBF values for Block RAMs

Mission	MTBF
Sentinel-3	4.6 days
Sentinel-3 (SPE)	13.7 minutes
Galileo	18.1 days
Galileo (SPE)	4.1 minutes

Table 22: Steady-state availability: No redundancy

Mission	Reconfiguration Interval (images)			
	5	50	500	5000
Sentinel-3	0.999998	0.99998	0.9998	0.998
Sentinel-3 (SPE)	0.9991	0.992	0.93	0.52
Galileo	0.9999993	0.999994	0.99994	0.9994
Galileo (SPE)	0.997	0.97	0.79	0.20

resulting MTBF figures can be found in Table 21 and express how long it takes in average until a failure occurs at the output of the stream processor that was triggered by an upset in one of the Block RAMs.

7.5 AVAILABILITY ANALYSIS RESULTS

7.5.1 *No Redundancy & Periodic Reconfiguration*

First, the steady-state availability of a stream processor is determined, to which no redundancy is applied. It is assumed that the images are processed in bursts, i.e. that a partial reconfiguration can be done between the bursts periodically. For modelling, the stochastic Petri nets for approach 1 from Section 7.5.1 are used. The first net, see Figure 29, is slightly simplified. Instead of the six exponential transitions `mod_seu_*`, only two are required now: The first transition is triggered using the MTBF value for the configuration memory, the second one is triggered using the MTBF value for the Block RAMs. For the deterministic transition `df_proc_time`, 100 ms is used since images in the proof of concept system are transmitted with a frequency of 10 frames per second. The reconfiguration interval threshold value `v_reconf_intvl` represents the burst length. For this case study, burst lengths of 5, 50, 500 and 5000 images are chosen. The

Table 23: Steady-state availability: DWC and TMR

Mission	DWC	TMR
Sentinel-3	0.999996	0.999999999...
Sentinel-3 (SPE)	0.998	0.9999998
Galileo	0.999998	0.999999999...
Galileo (SPE)	0.995	0.999998

results for the different mission scenarios and burst lengths are given in Table 22.

7.5.2 *Duplication with Comparison & On-Demand Reconfiguration*

If the stream processor is duplicated and the voter module is used in comparator mode, faulty stream processors can be repaired by the FDIR supervisor on demand. If a mismatch occurs between the network output streams of the JPEG stream processors, the comparator cannot determine which one of the processors is faulty and is thus marking both slots as being faulty. This change in the health status of the voter module triggers an interrupt in the FDIR supervisor software, which executes a partial reconfiguration of both stream processors. In the following, it is assumed that the failure detection time is 50 ms, the reconfiguration time for each processor is also 50 ms and the voter resynchronisation takes another 100 ms, leading to a total repair time of 200 ms. To model this FDIR configuration, the stochastic Petri net for approach 3, see Figure 34, is solved. The resulting availability figures for the different mission scenarios are given in Table 23.

7.5.3 *Triple Modular Redundancy & On-Demand Reconfiguration*

If the stream processor is triplicated, a faulty stream processor can also be repaired on demand as described above. To model this configuration, the stochastic Petri net for approach 4, see Figure 35, is solved. This time, however, one token represents one stream processor and as a result the repair time is the one required for only one processor instance. Thus, the total repair time is only 150 ms instead of 200 ms. The availability figures for the different mission scenarios are given in Table 23.

7.6 DISCUSSION

The steady-state availability is an excellent figure to compare different FDIR configurations against each other. As can be seen in Table 22, an approach solely based on periodic reconfiguration can lead to a decent stream processor availability if the reconfiguration interval is rather short. During solar particle events, however, the approach performs poorly. In addition, this approach is only of interest if the data is processed in bursts. Then, a partial reconfiguration can be executed during these bursts. It must also be kept in mind that the figures given here are describing the so-called *inherent* steady-state availability, i.e. planned downtime due to the periodic reconfiguration is not visible in the availability figure.

As can be seen in Table 23, the Duplication with Comparison mode performs well, even during solar particle events. Since on-demand reconfiguration is used, the downtime due to a failure is very low. In addition, this mode can be used for applications in which the data is processed continuously. Still, lost JPEG images must be accepted from time to time. If this is no option, the Triple Modular Redundancy approach can further increase the availability, making lost images an extremely rare event.

From this comparison it can be concluded that Duplication with Comparison seems to be a decent solution for many satellite payload data processing applications. Only during harsh mission phases, a switch to TMR mode might be necessary. This observation supports the idea behind the proposed Distributed Failure Detection method: Since the method is adaptive, redundancy can be added or removed during flight and therefore the system availability can be increased or decreased too, depending on the current radiation environment and other mission constraints.

7.7 ADVANTAGES OF BLOCK RAM PROFILING

As discussed in Section 4.3.2, the susceptibility of the Block RAM bits cannot be neglected in availability analysis of SRAM-based FPGAs due to the large amount of Block RAMs used in some satellite payload data processing applications. In order to substantiate this claim, the reliability of a stream processor, to which no failure recovery method is applied, will be analysed and compared for the following three cases:

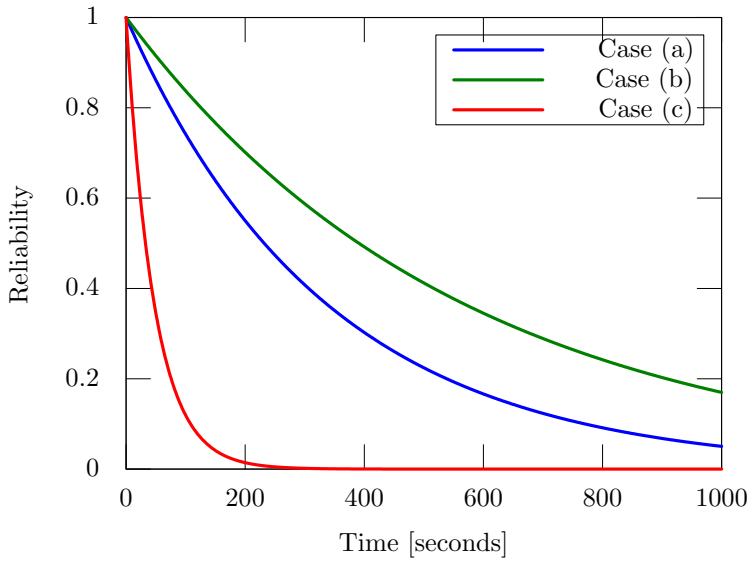

Figure 59: Influence of Block RAM profiling

- Case (a): The susceptible Block RAM bits, determined by the Block RAM profiling tool, are taken into account in the availability analysis.
- Case (b): The Block RAMs are fully neglected in the availability analysis.
- Case (c): All Block RAM bits within the stream processor are assumed to be susceptible in the availability analysis.

For illustration, the mission *Sentinel-3 (SPE)* is used in the ensuing analysis. For cases (a) and (b), the MTBF values are listed in Table 20 and Table 21. For case (c), where all Block RAM bits are assumed to be sensitive, the corresponding MTBF value can be calculated in the same way, but this time all 1,529,856 Block RAM bits must be taken into account, resulting in:

$$\begin{aligned}&\mathrm{MTBF}(F_{RAM/A}) = \\ &\quad (1.1 \cdot 10^{-3}\,\mathrm{SEU/bit{\cdot}day} \cdot 1529856\,\mathrm{bits})^{-1} \cdot 86400\,\mathrm{s/day} = 51.3\mathrm{s} \quad (7)\end{aligned}$$

Since the system becomes unreliable each time when one of the failure modes occurs, its reliability diagram can be represented by a

Table 24: Steady-state availability: Block RAM influence

Repair Interval (images)	Case (a)	Case (b)	Case (c)
50	0.992	0.997	0.95
500	0.93	0.97	0.62
5000	0.52	0.75	0.09

series connection of blocks corresponding to each failure mode. The reliability of a series system over time can be calculated as follows:

$$R(t) = e^{-t \sum_{i=1}^{n} \lambda_i} \tag{8}$$

where t is the time and $\lambda_i = 1/\text{MTBF}$ the failure rate of mode i.

The reliability over time of the aforementioned three cases is plotted in Figure 59. It can be seen from Figure 59 that the reliability curve corresponding to case (c), where all Block RAM bits are assumed to be sensitive, drops to zero very quickly. The difference of the prediction between case (b), where no Block RAMs are considered and case (c) can be as high as 75%. However, since the Block RAM profiling tool identifies only a fraction of Block RAM bits as sensitive, the curve of the corresponding case (a) is the more realistic estimate. The maximum difference of the prediction between case (a) and the other cases is 37%. From this example, it becomes clear that the impact of the susceptible Block RAM bits cannot be ignored. Without any knowledge about the usage of the Block RAMs, the actual reliability curve could lie anywhere between case (b) and (c).

It has to be noted that the degree to which the total number of the estimated sensitive Block RAM bits eventually affects the estimated availability figure depends very much on the failure recovery approach. As an example, Table 24 shows the availability figures computed for the approach, in which a single stream processor is periodically reconfigured (approach 1 in Section 4.4.2.1). If the stream processor is repaired very frequently, say every 5 images, the impact is rather low because at this point of time, the probability that the stream processor is still healthy is high for all three cases. However, if it is repaired less frequently, e.g. every 5,000 images, the impact becomes much more visible as evidenced by the bottom row in Table 24.

7.8 CONCLUSIONS

In this chapter, the proposed Availability Analysis method was practically applied to the proof of concept system. The JPEG stream processor was characterised using the proposed fault injection system and Block RAM profiling tool. It was proven that random fault injection can provide accurate results, which is a completely new finding. The MTBF and the availability figures were determined for several redundancy configurations, based on two example satellite missions. By doing so, it was shown for the first time that Duplication with Comparison and Triple Modular Redundancy can perform extremely well when coupled with on-demand reconfiguration. Furthermore, the positive influence of the Block RAM profiling tool was illustrated by comparing the predicted figures to two border cases, in which either the Block RAMs are not taken into account at all or assumed to be fully susceptible.

8

CHAPTER 8: VALIDATION OF FDIR STRATEGY

8.1 INTRODUCTION

With support from the European Space Agency, an accelerated proton irradiation test campaign was conducted at the Paul Scherrer Institut (PSI) in Villigen, Switzerland in the night from the 8th to the 9th of May 2015. Christian Poivey, staff at ESA-ESTEC, was in control of the proton beam while the author of this thesis was monitoring and controlling the device under test. The test started around midnight. Wojciech Hajdas from PSI set up and calibrated the beam, which took around three hours to finish. Unforeseen problems with secondary radiation effects further delayed the start of the actual test. Thus, the first successful test run was started around 4 am. Three experiments were conducted, each with a series of test runs. The last run was finished around 12 pm.

Main objectives of the test campaign were:

- The validation of the Distributed Failure Detection method, proposed in Chapter 3, by proving that the proof of concept system is able to automatically detect failures and recover itself in a real radiation environment using a real-world application design.
- The validation of the Availability Estimation method, proposed in Chapter 4, by proving that the estimated availability for the proof of concept system correlates with the system availability measured during the test campaign.

The chapter is structured as follows. In Section 8.2, the test setup is described in detail, including the test bench, the FPGA designs and the test procedure. Then, all test results are given in Section 8.2, including the outcome of the static and dynamic tests and the estimation of the MTBF and availability figures based upon the test results. Finally, Section 8.4 concludes the chapter by summarising the findings.

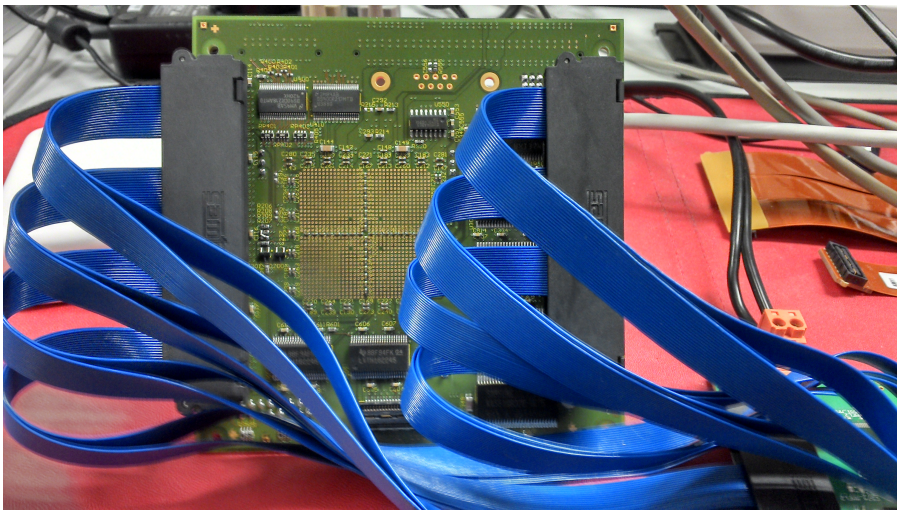

Figure 60: Backside of DRPM daughterboard

8.2 TEST SETUP

The DUT is a commercial FPGA device of type Virtex-4 XC4VSX55 by manufacturer Xilinx. The DUT is mounted on a daughterboard of the DRPM system with dimensions: 115 x 115 mm², see Figure 37. The DUT as well as all other components on the daughterboard are not radiation-hardened by design. No components are mounted behind the DUT on the backside of the daughterboard. Only passive components (mainly resistors and capacitors) are mounted in the very direct surrounding of the DUT. The daughterboard is connected to the DRPM mainboard via two high speed interconnects by manufacturer Samtec, see also Figure 60. To guarantee signal integrity, the length of the cables is only 10 inch (254 mm). However, the cables are flexible, allowing a certain degree of freedom in mounting the daughterboard during the test campaign.

Aside from the daughterboard with the mounted DUT, five other devices are located in the test chamber as can be seen in the block diagram in Figure 61:

- The DRPM mainboard is connected via a SpaceWire cable to an Ethernet-to-SpaceWire bridge by manufacturer 4Links.
- The DRPM mainboard is connected via two SpaceWire cables to the SPWRTC board.
- Both, the mainboard and the SPWRTC board are powered by a switchable power supply.

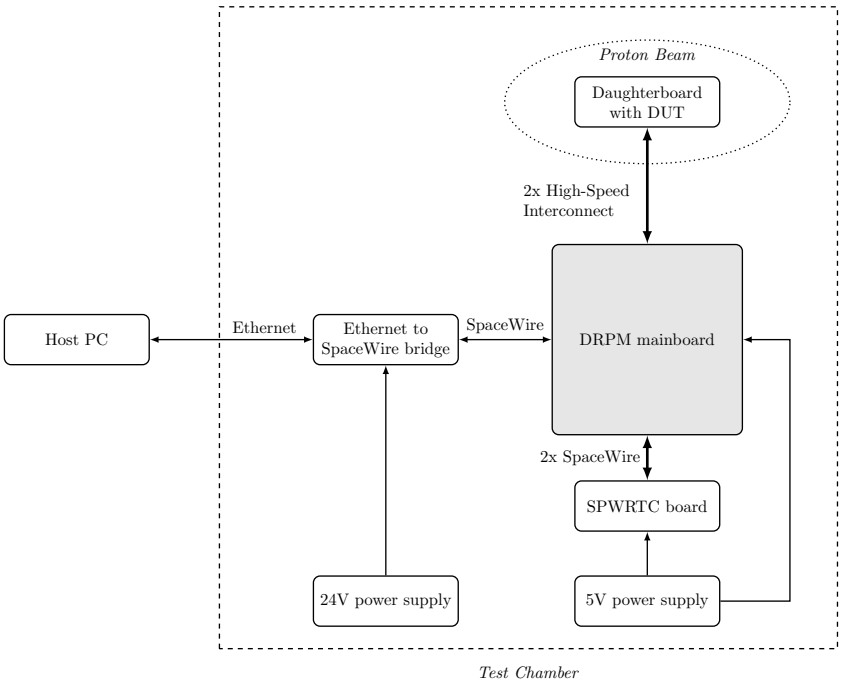

Figure 61: Block diagram of the test bench used during the beam test

- The Ethernet-to-SpaceWire bridge is connected via an Ethernet cable to the host PC, located in the control room.
- The Ethernet-to-SpaceWire bridge is powered by a 24 V desk power supply.

Figure 62 shows a view of the test chamber. The beam line is followed by a collimator (large metal ring) that shapes the proton beam. The DUT is mounted behind the collimator.

A simplified block diagram of the FPGA design that is used during the irradiation campaign is shown in Figure 63.

The first FPGA is mounted on the DUT board that is irradiated. The FPGA design was minimised as much as possible to simplify the analysis of the resulting data, e.g. the DCMs were removed as they are known to be prone to single event effects. As a proof of concept application, the JPEG image compression stream processor is used, which was described in Section 5.3.2. Since the proposed FDIR approach is based on dynamic partial reconfiguration, the stream processor is hosted on a reconfigurable partition.

Aside from the stream processor, only a NoC to NoC bridge is implemented on the irradiated FPGA, which is necessary to commu-

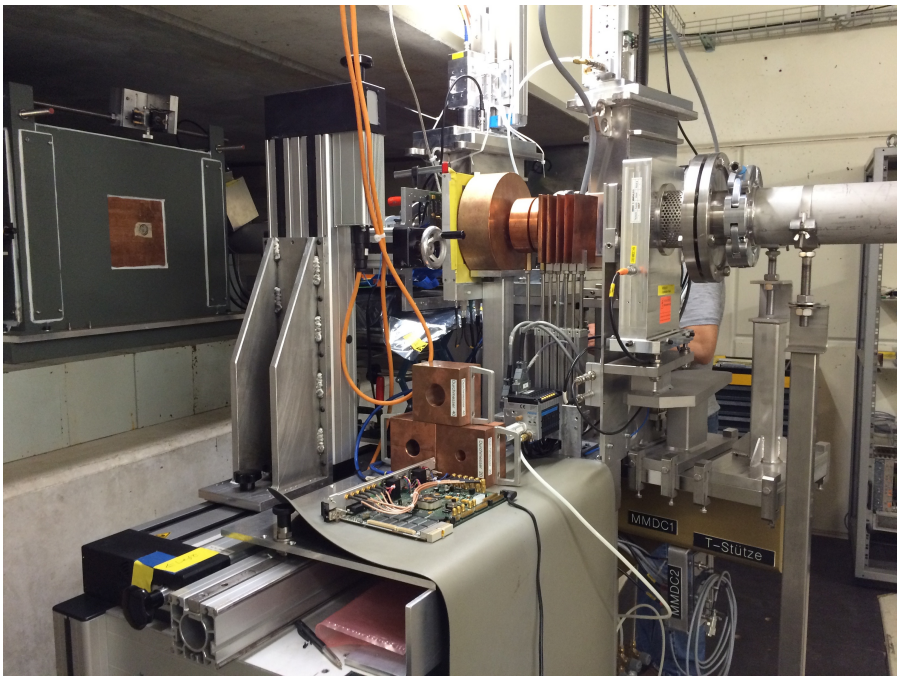

Figure 62: A view of the test chamber

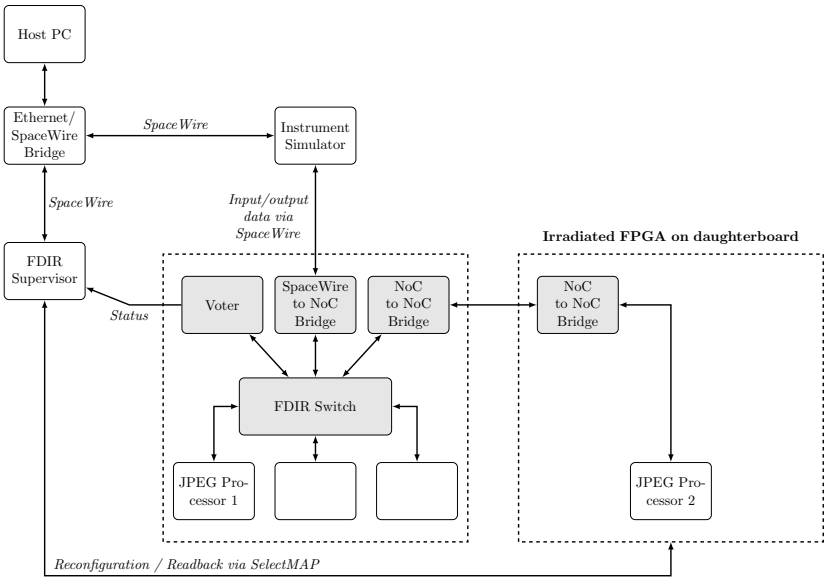

Figure 63: Block diagram of the FPGA design used during the beam test

nicate with the second FPGA. The cross-section of the bridge is not known but is assumed to be very small and is thus neglected.

The second FPGA is mounted on the mainboard that is not irradiated during the test campaign. It comprises a second JPEG stream processor, which works in hot redundancy together with the JPEG stream processor hosted on the irradiated FPGA. It further comprises a FDIR routing switch, to which both stream processors are connected. In addition, the NoC is bridged to a SpaceWire interface, which allows the communication with external components. The voter module within the routing switch is used as a comparator. Once this module detects a failure, a health status is flagged to the FDIR supervisor software running on the LEON3 microprocessor, which is implemented on the ProASIC3e FPGA on the same mainboard. The LEON3 microprocessor can access the configuration memory of both Virtex-4 FPGAs via their SelectMAP interface.

The following steps are repeated throughout the test campaign:

1. Every 100 milliseconds, the instrument simulator software (implemented on the SPWRTC board) sends a full raw image via SpaceWire to the second FPGA.

2. Within this FPGA, the raw image is multicast to both JPEG stream processors.

3. Both JPEG stream processors process the raw image and send the resulting JPEG image to the comparator module.

4. The comparator module is doing a bitwise comparison of the two redundant network streams. In case of a mismatch, the comparator module flags a health status to the FDIR supervisor software and stops forwarding any data, i.e. the comparator module is fail-silent. The following steps are initiated after a failure detection:

    - The FDIR supervisor reads back the configuration bitstream of the irradiated FPGA for later data analysis.
    - Since the FDIR supervisor cannot determine which JPEG stream processor failed, both stream processors are reconfigured via their SelectMAP interfaces.
    - After reconfiguration, the FDIR supervisor sends a request to the comparator module to continue the comparison.

5. If no mismatch occurs, the comparator module is simply forwarding the JPEG image to the SPWRTC board via SpaceWire.

6. The instrument simulator software running on the SPWRTC board is comparing the JPEG image to a "golden copy". A counter is keeping track of the number of correctly received JPEG images. At the same time, another counter is keeping track of the number of transmitted raw images. Therefore, the software can continuously determine the availability of the system.

8.3 TEST RESULTS

8.3.1 *Overview*

The proton test was conducted at two energy levels: 100 MeV (test runs 17 to 21) and 200 MeV. At 200 MeV, two different fluxes were used, ca. $4.2 \cdot 10^6$ p/cm²·s (test runs 3, 4 and 7 to 12) and ca. $8.3 \cdot 10^6$ p/cm²·s (test runs 5, 6 and 13 to 16). While the test was running, a radiation detector measured the fluence [p/cm²]. Since the active beam time was measured too, the flux can easily be calculated by dividing the measured fluence by the run time.

It turned out that the shielding of the two FPGAs, which were assumed to be reliable (second Virtex-4 and ProASIC3e device), was not sufficient. During the test, SEUs also occurred in these devices from time to time, most likely due to neutron scattering. Two basic failure modes were observed that indicated upsets in these devices: (i) the ProASIC3e device stopped the transmission of status information to the host PC and (ii) the SPWRTC board did not receive any images back from the second Virtex-4 FPGA. To circumvent this unforeseen issue, the DRPM system was connected to a power supply that was switchable from the control room. By doing so, the test could be manually stopped every time one of the aforementioned failure modes occurred and the DRPM system was power cycled. Then, the system was set up again and the test restarted. Therefore, the test result data presented in the following was gathered from several test runs. The first two test runs no. 1 and no. 2 are not taken into account since the aforementioned issue was detected during these runs and thus the gathered results must be assumed to be (partly) wrong. Also test run no. 8 was skipped because a power cycle was already necessary after the detection of two failures, i.e. the sample size was too small to be taken into account for further data analysis.

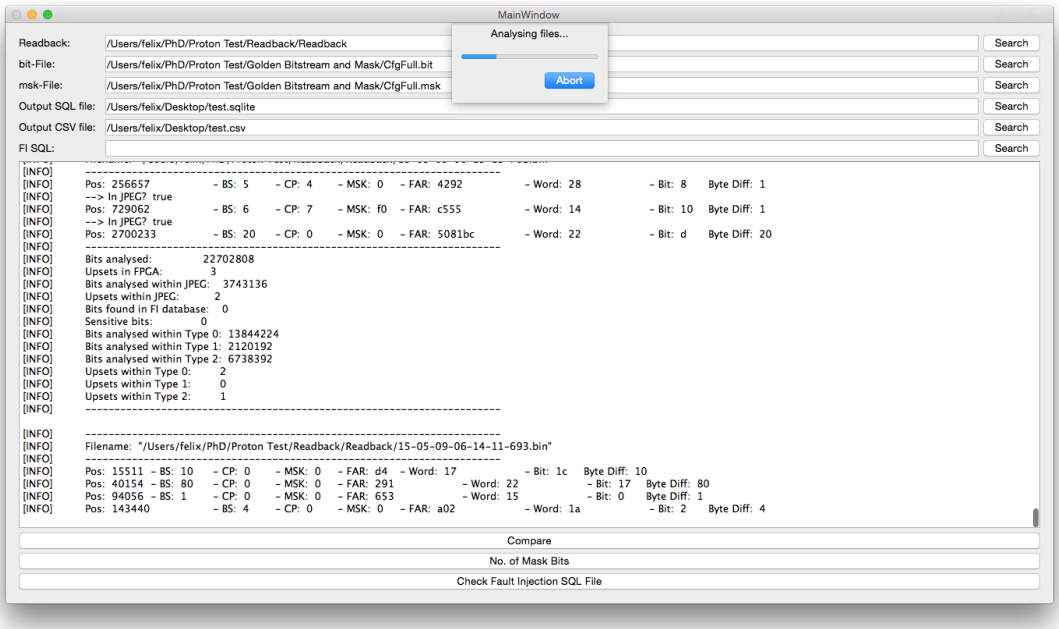

Figure 64: Screenshot of bitstream compare tool

8.3.2 *Static SEU Characterisation*

After each failure detection, the bitstream was read back from the Virtex-4 device, transmitted to the host PC via SpaceWire and stored to hard disk. A custom-built tool was developed for post analysis of all bitstreams. A screenshot of the tool is shown in Figure 64. First, the readback bitstreams are aligned to a golden bitstream (*.bit file) and a masking file (*.msk file) generated by the Xilinx toolchain. Then, the files are compared byte-wise. To do so, the readback file is XORed with the bitstream file and the inverted mask file is applied by an AND operation. The resulting byte contains logical 1s at the positions where an upset occurred. If the byte is not zero, the algorithm steps through all bits of the byte to identify the exact bit position of the upset. For this bit position, the FAR address of the corresponding frame is determined. Since the tool is aware of the FPGA's internal memory structure, it is able to determine if the upset occurred in the area of the JPEG stream processor, in Type 0, Type 1 or Type 2 blocks. Type 0 blocks comprise CLBs, IOBs and DSPs. Type 1 blocks comprise bits responsible for Block RAM interconnect. Type 2 blocks comprise the Block RAM bits.

Each bitstream that was read back from the device contains several SEUs. This is due to the fact that many bits are not used in the

Table 25: Overview: Number of SEUs in different memory blocks

Experiment	SEUs JPEG	SEUs all FPGA	SEUs Type 0	SEUs Type 1	SEUs Type 2
200 MeV 4.2E+06	2606	16938	9244	1484	6210
200 MeV 8.3E+06	1796	11762	6509	963	4290
100 MeV 8.5E+06	1970	12860	6891	1161	4808
All	6372	41560	22644	3608	15308

design and that it therefore takes some time until a sensitive bit is hit by a particle, which eventually causes a measurable failure. In the meanwhile, several SEUs accumulate in the configuration memory. We use this fault accumulation to our advantage and calculate the static cross-sections of the device with it. Since the aforementioned tool can distinguish between Type 0, 1 and 2 blocks, separate cross-sections for both the CLBs and the Block RAMs can be calculated. Memory content of Block RAMs actually used by the design cannot be analysed as it is masked out by the masking file. However, around two-thirds of the Block RAMs are not used by the design and were thus available as on-chip radiation detectors. An overview of the number of detected SEUs for the different configuration memory blocks can be found in Table 25, a graphical representation is shown in the pie charts in Figure 65.

The aforementioned bitstream analysis tool also determined the number of bits in the different blocks of the golden bitstream file and the masking file, the results can be found in Table 26. In the following, the number of comparable bits is of interest, which is the number of configuration bits minus the number of masking bits.

Using the figures from Table 25 and 26, the cross-section of a specific memory block type can easily be determined by dividing the number of SEUs by the measured fluence. Then, the cross-section per bit is calculated by dividing the cross-section value of the memory block by the number of comparable bits within this memory block.

To validate the Availability Analysis method proposed in the course of this PhD, see Section 4, two cross-sections are of particular interest. The first cross-section covers the configuration memory bits (CLBs, IOBs, DSPs, Block RAM Interconnect etc.) whereas the second cross-

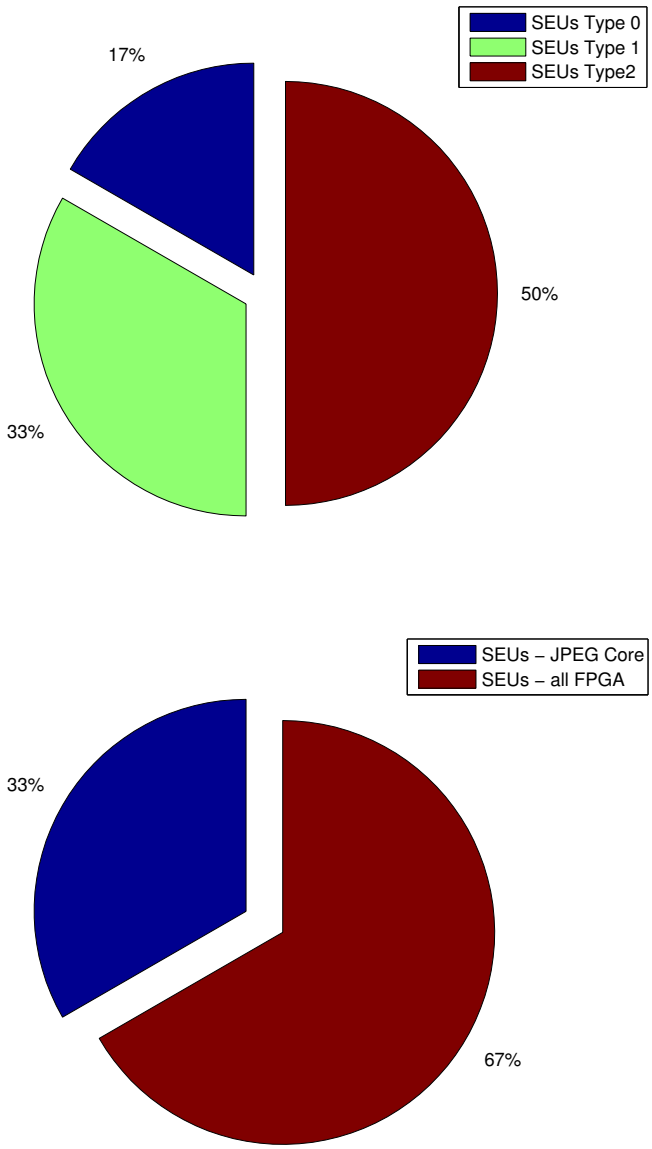

Figure 65: Percentage of detected SEUs in different memory blocks

Table 26: Number of bits in golden bitstream file and masking file

Block	Configuration Bits	Masking Bits	Comparable Bits
Total FPGA	22702808	2936184	19766624
JPEG	3873024	153072	3719952
Type 0	13844224	539344	13304880
Type 1	2120192	74240	2045952
Type 2	6738392	2322600	4415792

Table 27: Cross-Section: Configuration Memory (Type 0 and Type 1)

Experiment	Fluence [p/cm²]	Runtime [s]	SEUs	X-Section [cm²]	X-Section per bit [cm²/bit]
200 MeV 4.2E+06	3.82E+10	9176	10728	2.81E-07	1.83E-14
200 MeV 8.3E+06	2.67E+10	3207	7472	2.80E-07	1.83E-14
100 MeV 8.5E+06	3.42E+10	4028	8052	2.36E-07	1.54E-14

Table 28: Cross-Section: Block RAM (Type 2)

Experiment	Fluence [p/cm²]	Runtime [s]	SEUs	X-Section [cm²]	X-Section per bit [cm²/bit]
200 MeV 4.2E+06	3.82E+10	9176	6225	1.63E-07	3.69E-14
200 MeV 8.3E+06	2.67E+10	3207	4290	1.61E-07	3.64E-14
100 MeV 8.5E+06	3.42E+10	4028	4808	1.41E-07	3.19E-14

section is for the Block RAM memory cells. For all experiments, both cross-sections can be found in Table 27 and 28.

For comparison, the cross-sections for the configuration memory and the Block RAMs in (space graded) XQR4VSX55 devices, provided by NASA and Xilinx [3], can be consulted. However, as shown in Figure 56 and Figure 57, the device was only characterised up to 60 MeV. Still, it can be seen that our measured cross-sections for 100 MeV and 200 MeV are plausible as they are following the trend quite well, although the Weibull fits seem to be too conservative, presuming slightly larger cross-sections.

8.3.3 *Dynamic Testing: Experiment No. 1*

Test runs 3, 4 and 7 to 12 were conducted at 200 MeV with a flux of ca. $4.2 \cdot 10^6$ p/cm²·s. The overall run time was 9176 s and 439 failures

Table 29: Overview: 200 MeV, $4.2 \cdot 10^6$ p/cm²·s

Run	Fluence [p/cm²]	Run Time [s]	Flux [p/cm²·s]	# Failures
3	3.70E+09	889	4.16E+06	49
4	1.00E+10	2410	4.15E+06	118
7	3.84E+09	910	4.22E+06	44
9	4.10E+09	991	4.14E+06	44
10	1.16E+09	275	4.22E+06	13
11	6.80E+09	1639	4.15E+06	63
12	8.60E+09	2062	4.17E+06	108
Total:	3.82E+10	9176	4.16E+06	439

were detected in total. Table 29 gives an overview of the individual test runs.

After each failure detection, the FDIR supervisor software transmitted a status message to the host PC, which contained the time that elapsed since the last failure detection. As expected, the probability distribution of these time to failure values can be approximated by an exponential distribution as can be seen in the histogram in Figure 66. Averaging these time values results in the MTBF figure for a specific test run. An overview of the calculated MTBF values can be found in Table 30.

Table 30: MTBF Values: 200 MeV, $4.2 \cdot 10^6$ p/cm²·s

Run	MTBF [s]
3	17.15010417
4	19.71884615
7	20.2452907
9	21.28947674
10	20.46020833
11	24.83907258
12	19.03834112
Overall mean:	20.22905671

The averaged MTBF value can now be compared to the predicted MTBF value that is based on our estimation of sensitive configuration memory and Block RAM elements. Therefore, this most crucial

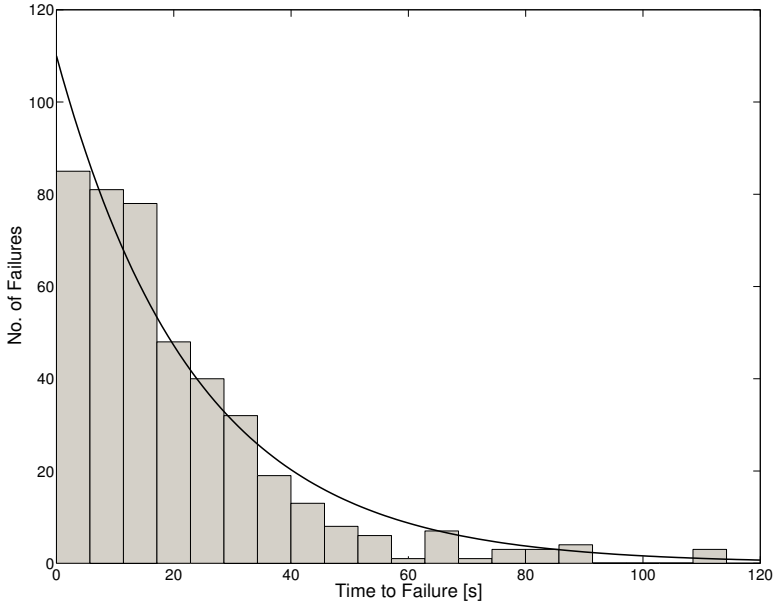

Figure 66: Time to failure values of the first 200 MeV experiment

part of the proposed availability analysis method can indirectly be validated.

The Block RAM profiling tool determined 67,858 sensitive Block RAM bits used as RAMs and 27,351 sensitive Block RAM bits used as ROMs, see Section 7.4.2. The random fault injection campaign predicted around 526,300 sensitive configuration memory bits within the JPEG stream processor, see Section 7.4.1. Using the measured cross-sections, the MTBF value is predicted as follows:

$$\begin{aligned} \text{MTBF} = [4.16 \cdot 10^{6}\,\text{p/cm}^2\text{·s} \cdot (1.83 \cdot 10^{-14}\,\text{cm}^2\text{/bit} \cdot 526300\,\text{bits} + \\ 3.69 \cdot 10^{-14}\,\text{cm}^2\text{/bit} \cdot (67858 + 27351)\,\text{bits})]^{-1} = 18.275\,\text{s/p} \end{aligned} \quad (9)$$

As can be seen, the error of the prediction is low. In relation to the measured value of 20.23 s, the predicted value of 18.28 s is off by only 9.7%.

8.3.4 *Dynamic Testing: Experiment No. 2*

Test runs 5, 6 and 13 to 16 were conducted at 200 MeV with a flux of ca. $8.3 \cdot 10^{6}$ p/cm²·s. The overall run time was 3207 s and 343 failures

Table 31: Overview: 200 MeV, $8.3 \cdot 10^6$ p/cm²·s

Run	Fluence [p/cm²]	Run Time [s]	Flux [p/cm²·s]	# Failures
5	1.70E+09	203	8.37E+06	2
6	2.80E+09	333	8.41E+06	37
13	8.22E+09	994	8.27E+06	91
14	2.02E+09	246	8.21E+06	28
15	1.92E+09	231	8.31E+06	27
16	1.00E+10	1200	8.33E+06	140
Total:	2.67E+10	3207	8.31E+06	343

Table 32: MTBF Values: 200 MeV, $8.3 \cdot 10^6$ p/cm²·s

Run	MTBF [s]
5	9.822631579
6	8.253333333
13	10.69113889
14	8.05037037
15	8.660288462
16	8.59028777
Overall mean:	9.146973294

were detected in total. Table 31 gives an overview of the individual test runs.

The calculated MTBF values for all test runs can be found in Table 32. Again, the averaged MTBF value can be compared to the predicted MTBF figure:

$$\begin{aligned} \text{MTBF} = [8.31 \cdot 10^6 \text{p/cm}^2\text{·s} \cdot (1.83 \cdot 10^{-14}\ \text{cm}^2/\text{bit} \cdot 526300\,\text{bits} + \\ 3.64 \cdot 10^{-14}\ \text{cm}^2/\text{bit} \cdot (67858 + 27351)\,\text{bits})]^{-1} = 9.185\,\text{s/p} \end{aligned} \quad (10)$$

This time, the predicted value of 9.185 s matches the measured value of 9.147 s nearly perfectly, with an error of only 0.4%.

Table 33: Overview: 100 MeV, $8.5 \cdot 10^6$ p/cm²·s

Run	Fluence [p/cm²]	Run Time [s]	Flux [p/cm²·s]	# Failures
17	8.99E+09	1100	8.17E+06	81
18	2.83E+09	330	8.58E+06	24
19	5.76E+09	669	8.61E+06	44
20	1.00E+10	1164	8.59E+06	92
21	6.57E+09	765	8.59E+06	68
Total:	3.42E+10	4028	8.48E+06	309

Table 34: MTBF Values: 100 MeV, $8.5 \cdot 10^6$ p/cm²·s

Run	MTBF [s]
17	12.9291875
18	13.61826087
19	14.05593023
20	12.16035714
21	11.14865672
Overall mean:	12.51813322

8.3.5 *Dynamic Testing: Experiment No. 3*

Test runs 17 to 21 were conducted at 100 MeV with a flux of ca. $8.5 \cdot 10^6$ p/cm²·s. The overall run time was 4028 s and 309 failures were detected in total. Table 33 gives an overview of the individual test runs and the calculated MTBF values can be found in Table 34.

The averaged MTBF value is again compared to the estimated MTBF value:

$$\begin{aligned} \text{MTBF} = [8.48 \cdot 10^6 \text{p/cm}^2\text{·s} \cdot (1.54 \cdot 10^{-14} \text{ cm}^2\text{/bit} \cdot 526300 \text{ bits} + \\ 3.19 \cdot 10^{-14} \text{ cm}^2\text{/bit} \cdot (67858 + 27351) \text{ bits})]^{-1} = 10.586 \text{ s/p} \end{aligned} \quad (11)$$

For this experiment, the prediction error is 15.4% and is thus slightly greater than the error figures seen in the previous experiments.

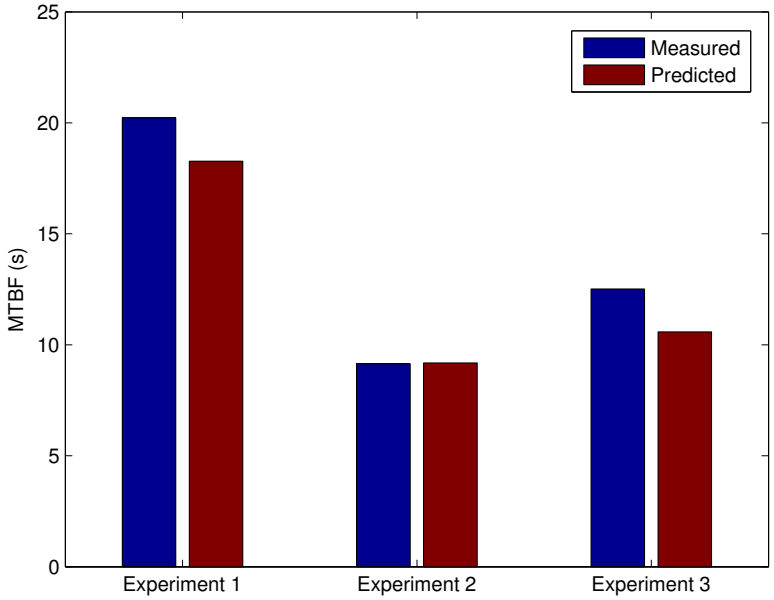

Figure 67: Measured versus predicted MTBF values

Table 35: Measured availability during test campaign

Experiment	TX Images	RX Images	Availability
200 MeV			
4.2E+06	80724	77621	0.961560
200 MeV			
8.3E+06	28586	26266	0.918841
100 MeV			
8.5E+06	35845	33741	0.941303

A graphical comparison of the measured and the predicted MTBF values for all three aforementioned experiments can be found in Figure 67.

8.3.6 *Availability Analysis*

During the beam test, the instrument simulator software was counting the transmitted images and the correctly received images. Thus, the availability could simply be calculated by dividing the number of correctly received images by the number of transmitted images. The results are given in Table 35.

Table 36: Predicted availability using stochastic Petri nets

Experiment	Availability *using measured MTBF*	Error [%]	Availability *using predicted MTBF*	Error [%]
200 MeV 4.2E+06	0.96388	0.2	0.96017	0.1
200 MeV 8.3E+06	0.92347	0.5	0.92377	0.5
100 MeV 8.5E+06	0.94290	0.2	0.93318	0.9

Using the measured MTBF values given in the previous sections, the steady-state availability can also be predicted using stochastic Petri nets. Since Duplication with Comparison was used as redundancy scheme, the Petri net shown in Figure 34 is used. Instead of multiple exponential transitions `mod_seu_x`, only one transition is necessary, which is triggered with the averaged MTBF value for each test experiment. Two more parameters are needed: The exponential detection time `mod_detect` and the deterministic repair time `mod_rep`. For the detection time, an average value of 50 ms is assumed, which corresponds roughly to the processing time of one image. The repair time can be determined more precisely. During the beam test, the FDIR supervisor software measured the time span between the failure detection and the successful update of the voter health status after repair. In average, the failure recovery procedure took 608 ms, including the readback of the bitstream. To this figure, another 100 ms must be added, which corresponds to the image frame period. This is required because the first image after repair is always corrupted due to the technical implementation of the voter module and the stream processor. The freshly repaired stream processor starts data processing only at the beginning of a new full image. Then, this new image is processed by the voter module. However, the voter module needs at least one network packet to reintegrate the freshly repaired processors. Thus, the first JPEG image after repair becomes corrupted since this network packet, which comprises the first 1 kB data block of this JPEG image, is skipped by the voter module.

The availability can also be predicted using the MTBF values that were estimated in the previous sections (see Equations 9 to 11).

The results of the availability prediction for both cases are listed in Table 36. If the measured MTBF value is used, the predicted availabil-

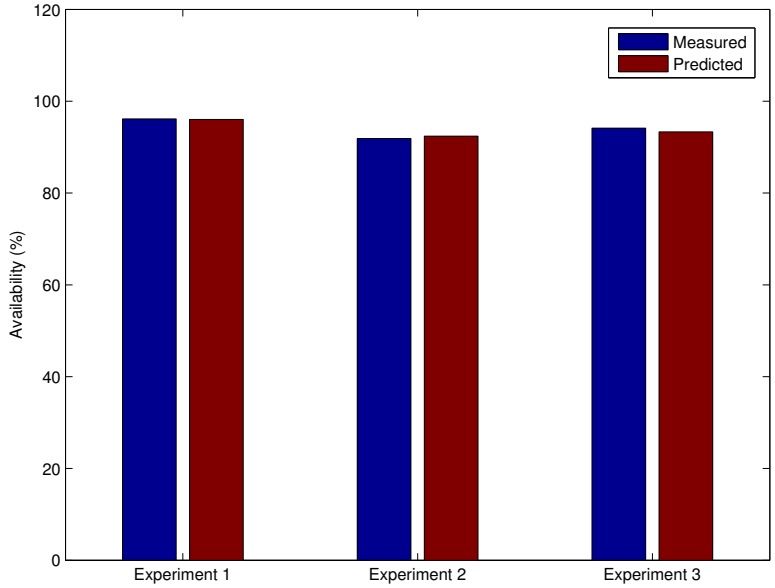

Figure 68: Measured versus predicted availability figures (based on predicted MTBF)

ity matches the measured availability extremely well with an error of 0.5% or less. This result shows that the stochastic Petri net model is well designed, although it must be pointed out that it delivers slightly too optimistic results.

If the predicted MTBF value is used instead, the predicted availability matches the measured availability nearly as well, which can also be seen in the bar chart in Figure 68. This result is very positive as it shows that the quantification of sensitive memory elements does not need to be extremely precisely. Only based on the fault injection experiments and the Block RAM profiling method, the estimated MTBF values were off by up to nearly 16% compared to the measured values. However, since the MTBF is so much longer than the average repair time, the error of the MTBF estimation plays only a negligible role for the availability estimation. This outcome strongly supports the idea of the availability analysis method proposed in this PhD work, which claims that a very good availability estimation is feasible with a rather simple and coarse-grain approach to the quantification of sensitive memory elements.

8.4 CONCLUSIONS

The accelerated proton irradiation campaign was a full success. Firstly, it was demonstrated that the FDIR hardware and software components are mature enough to detect and recover from failures in a real radiation environment that causes higher SEU rates than any solar particle event observed in history. Secondly, it was indirectly proven that the estimation of the number of sensitive bits (via fault injection experiments and Block RAM profiling) was quite accurate. It was possible to predict the MTBF value measured during the beam test with a maximum error of only 15.4%. Even more interestingly, it was shown that despite this error a much better prediction of the availability was feasible with an error of only 0.9% or less. Therefore, both the proposed Distributed Failure Detection and Availability Analysis method were successfully validated.

9

CHAPTER 9: CONCLUSIONS AND FUTURE WORK

9.1 SUMMARY

SRAM-based FPGAs are very interesting for modern approaches to data processing on board spacecraft since these devices offer high performance and a large amount of logic and embedded memory resources. However, except for the radiation-hardened Virtex-5QV device, all modern SRAM-based FPGAs suffer from SEU-induced failures in their configuration memory and user memory elements, regardless of their qualification level (space, defence or commercial). Hence, techniques are required, which can mitigate radiation-induced soft errors.

The thesis began with a thorough survey of mitigation techniques for SRAM-based FPGAs in Chapter 2. Since no recent work existed, which surveyed this rich research field, the literature survey on its own became an original contribution to knowledge in this field and was therefore later published in a leading computing survey journal. After a short introduction to relevant radiation effects, a common terminology was introduced to simplify the classification of the reviewed mitigation techniques. This was an important step since it turned out that one of the main obstacles for a good literature review is the heterogeneity of terminology used in scientific publications. Using this common terminology, the reviewed techniques could be classified as either being aimed at run time failure tolerance or at design time fault avoidance. Failure masking, detection and recovery techniques active during operation fall into the first category, whereas all techniques used during the design phase fall into the second category. The literature survey exposed two basic mitigation concepts: While the first one is based on netlist TMR combined with configuration memory scrubbing, the second one is based on modular redundancy combined with some on-demand failure recovery approach.

Since this PhD thesis is mainly aimed at high performance satellite data processing applications, the focus was from the beginning on a right balance of performance, reliability and power consumption, rather than solely on high reliability. A striking number of publica-

tions are only concerned with high reliability and often the motto seems to be: "The more the merrier". This is particularly true for the netlist TMR approach, which offers excellent reliability but comes with a high power, area and performance overhead. For payload data processing applications that are often based on complex algorithms, however, performance might be the number one factor and power and area is usually in short demand. Therefore, the second mitigation concept based on modular redundancy seemed to be the more promising candidate. In terms of performance, it was assumed that such an approach is superior to netlist TMR because less additional routing is required, which would eventually increase the critical path length of the circuit. In terms of power and area overhead, modular redundancy is particularly interesting in combination with dynamic partial reconfiguration. Then, hardware components, respectively redundant copies of these components, can be added and removed during operation, depending on the current power and area constraints of the space mission phase. As a consequence, the reliability of the system can be increased only if it is really necessary.

Although a good number of works in the field of modular redundancy exists, it turned out that most proposals are rather technology- and application-dependent and none of them is really applicable to multi-FPGA systems or at least scales well with the size of the application. This research gap was filled with the proposal of the novel Distributed Failure Detection method in Chapter 3. First, the application is partitioned into several hardware tasks, which are implemented within so-called stream processors. Regardless of the underlying algorithms, every stream processor has the same unified interface, which greatly simplifies the analysis of possible failures modes and decreases the hardware complexity of the required failure detectors. The stream processors are implemented on reconfigurable partitions, i.e. redundant copies can be added or removed during operation to adapt the system to current power or reliability constraints. The main novelty and the biggest difference to all other systems proposed in literature lies in the fact that the intercommunication between the stream processors is based on a switched fabric NoC rather than a communication bus or some kind of hard-wiring. Now, the communication scales well with the size of the application and the stream processors can be distributed over an arbitrary number of FPGAs in a multi-FPGA system. However, the question raised how FDIR methods can now be incorporated into such a communication archi-

tecture. The proposed Distributed Failure Detection method solves this problem by making failure detection mechanisms part of the network architecture. A voter module is embedded into a NoC routing switch, which can receive network streams from redundant stream processors, regardless of their position within the network topology. A FMEA revealed that the implementation of the voter module must be more complex than the implementation of a simple majority voter, because (i) the redundant network streams are not synchronous and (ii) failure modes specific to the network approach must be taken into account, for instance missing packets or so-called babbling idiots. The FMEA led to a unique, elegant and very small voter module design that can handle all failure modes and that can automatically switch from voter mode to comparator mode if one of the redundant network streams is missing. On the input side, data must be provided to the redundant stream processors simultaneously. This is solved by a simple but robust multicast mechanism that is also part of the NoC routing switch. Since network packets can be falsified due to failures in the stream processors, they can potentially congest the network if they take wrong routes. To avoid any congestion, failure isolation mechanisms were also added to the NoC routing switches. Finally, an idea for a possible data resynchronisation mechanism was proposed as well, which makes use of the FDIR components already available in the network. Often, the problem of data resynchronisation after repair is mentioned as the main drawback of all modular redundancy approaches.

Although the possibility of using different redundancy configurations is an attractive feature of the Distributed Failure Detection method, the question remains, what level of reliability can be achieved with each configuration. Answering this question is interesting (i) to compare different FDIR approaches against each other and (ii) to predict the achievable reliability in a specific radiation environment, i.e. on a specific space mission. The literature survey revealed that not much work regarding availability analysis for SRAM-based FPGAs exists. And the work that does exist is mainly making use of such analysis techniques to emphasise the advantages of a particular mitigation technique. Therefore, a novel Availability Analysis method is introduced in Chapter 4. While the workflow of the method itself might be not particularly novel, the incorporated techniques are innovative. The first problem of modular redundancy that must be taken into account is the fact that data in user memory is not automati-

cally resynchronised after repair as it is the case with a classic TMR and scrubbing approach. Therefore, faults in the configuration memory can manifest themselves as failures in the user logic if they get trapped in a feedback loop. Many failure recovery approaches exist, for instance partial reconfiguration or scrubbing, but not all of them can handle all possible failure modes. For example, a trapped failure in the user logic can only be repaired by means of partial reconfiguration or scrubbing combined with a circuit reset. To enable availability modelling for all possible combinations, a better understanding of the fault-failure mechanism is required, i.e. it must be possible to determine the probability of different failure modes, depending on how the system can recover from a failure. For example, it might be interesting to know the probability of failures that can be repaired by a simple scrub or the probability of failures that can only be repaired by a partial reconfiguration. This is achieved by a novel fault injection algorithm. In contrast to other approaches described in literature, it is not only able to determine the number of sensitive configuration memory bits but also the number of sensitive configuration memory bits that fall into a specific failure mode class. Another significant difference to other availability analysis methods described in literature is the fact that the proposed one also takes the influence of Block RAMs into account. Except for the number of Block RAMs utilised by a stream processor, not much is known about them when only the standard FPGA toolchain is used. Thus, a designer might decide to fully neglect the Block RAMs in an availability analysis or to assume that all Block RAM bits are susceptible. While this might be fine for a worst case analysis, it can be far off from reality, especially in systems with huge amounts of Block RAMs. To get a better estimate of the real susceptibility of the Block RAMs, an innovative Block RAM profiling tool was proposed. The idea is simple: A typical user case of a stream processor is simulated, e.g. for the processing of one single data block. During simulation, all read and write accesses to all Block RAMs are monitored. By doing so, it can be determined how many memory rows are actually used in a real-world scenario. But furthermore, the tool also takes into account the time spans in which a fault in a memory row cannot manifest itself as a failure because the content of the memory row is overwritten before the next read operation. By means of these time spans, a correction factor can be calculated that further increases the prediction precision. Finally, the proposed Availability Analysis method also comprises several exam-

ple stochastic models that are based on Petri nets. By solving these models, the availability of a particular FDIR configuration in a particular radiation environment can be calculated.

All FDIR components and availability analysis tools were implemented in a complex proof of concept system described in Chapter 5 that comprises hardware, embedded software and workstation software components. Most importantly, the system proves that the proposed mechanisms can actually be implemented in real hard- and software. In addition, the system became an excellent test bench for further research and validation purposes. The relevance of the proof of concept system is underlined by the fact that it comprises components very similar to the ones found in spaceborne hard- and software.

The first research activity carried out with the proof of concept system was concerned with power, area and performance measurements of different FDIR configurations, see Chapter 6. Most notably, it was shown that modular TMR performs better than netlist TMR in terms of performance and power overhead, a great result supporting the choice of modular TMR for high performance payload data processing applications. In addition, it was shown that failure recovery techniques can bear a large power overhead, which suggests that on-demand failure recovery approaches are more power efficient than fast, periodic ones.

The second research activity that made use of the proof of concept system is described in Chapter 7 and applied the Availability Analysis method to the JPEG stream processor that is implemented as part of the system. The availability of the stream processor in three different redundancy configurations was determined for two example space missions. As an important finding, it was proven that random fault injection delivers good results, making a full fault injection campaign unnecessary. Furthermore, it was shown that on-demand failure recovery approaches are superior to periodic ones in terms of achievable availability because the mean time to repair is minimised. Using one of the example space missions, the positive influence of the Block RAM profiling tool was demonstrated as well.

The third and most important experimental research activity is described in Chapter 8. Due to great support from the European Space Agency, it became possible to validate both the Distributed Failure Detection and the Availability Analysis method by means of accelerated proton testing. First of all, the test date became an important

deadline for a flawless implementation of the proof of concept system. The FDIR components of the Distributed Failure Detection method were all working as expected throughout the whole test campaign, i.e. all failures were successfully detected by the voter module and the system could always be recovered from the failure. The system was slightly simplified for the test campaign to ensure that the resulting data could be better analysed. Thus, only one redundancy configuration (Duplication with Comparison) was tested at two different energy levels. By doing so, the MTBF and availability values of the system could easily be measured in real-time and later compared to the figures predicted by the Availability Analysis method. It turned out that the measured MTBF values matched the predicted MTBF values very well, with a maximum error of only 15.4%. Thus, it was indirectly proven that the estimation of the number of sensitive bits (via fault injection experiments and Block RAM profiling) was quite accurate. The cross-sections required for the prediction of the MTBF values were calculated from the bitstreams that were read back from the irradiated FPGA device after each failure detection.

9.2 CONCLUSIONS

In this thesis, an innovative FDIR strategy for SRAM-based FPGAs is proposed, which is, in contrast to earlier work, specifically aimed at high performance satellite data processing applications. The rationale of this decision is that such high performance devices are best utilised in applications in which large data streams must quickly be processed in realtime, for instance in data compression or image feature extraction applications.

The proposed FDIR methodology comprises a hardware framework that interconnects so-called stream processors in a NoC architecture. Each stream processor executes one processing step in hardware and a data processing pipeline can easily be implemented by connecting several stream processors in series. Since the NoC is based on a modern switched fabric architecture this approach is unique in comparison with previous work. The processing pipeline and the functionality of its stream processors can arbitrary be changed during operation by reprogramming the routing tables and by utilising the dynamic partial reconfiguration feature of SRAM-based FPGAs. Furthermore, the NoC can span several FPGAs via off-chip intercon-

nections, which offers excellent scalability for high performance data processing applications.

Failure detectors are integrated into the NoC to recover from soft errors caused by radiation effects. This is a novel approach because the monitored stream processors are not hardwired to the detector(s). Instead, the output of any stream processor can be routed to any input of any failure detector in the network. By doing so, stream processors can be added, removed, duplicated, or relocated and then integrated into the failure detection process by simply reprogramming the routing tables of the network. It does not even play a role on which FPGA the failure detectors and the individual monitored stream processors are located. The proposed FDIR mechanisms were evaluated in terms of power and area overhead as well as performance. It was shown that they have no significant disadvantages compared to other approaches. Rather, the analysis of an example application revealed that the performance of a circuit can be increased when using the proposed FDIR methodology instead of a classic TMR approach that is applied to the netlist of the circuit.

As a proof of concept, the hardware framework was successfully implemented in a complex demonstration system that resembles a real flight system. The correct functionality was validated in a proton irradiation test campaign. It was shown that the FDIR methodology can withstand much higher soft error rates than the ones that must be expected during space missions.

To estimate the reliability of the data processing pipeline, a new availability analysis method is proposed in this thesis as well. The number of sensitive SRAM configuration memory bits is predicted by a random fault injection system, whereas the number of sensitive Block RAM memory bits is predicted by a novel Block RAM profiling algorithm that analyses memory accesses during a simulation run. The FDIR configuration is modelled by stochastic Petri nets and the availability figures are determined by solving the underlying Markov chains. Compared to earlier work, the proposed approach does not neglect the influence of Block RAM bit upsets and thus provides more accurate results. Furthermore, it also allows the modelling and analysis of more complex FDIR configurations as it is based on Petri nets.

The availability analysis was also validated during the proton irradiation test campaign. Beforehand, the error rates expected during the test were predicted for the demonstration system using the proposed analysis method. It was shown that both the predicted error

rates and the resulting availability figures match the measured figures accurately. In addition, the test provided new data not yet published in other work, which enabled the calculation of the static cross-section for Virtex-4 SX55 devices at energies of 100 MeV and 200 MeV.

9.3 CONTRIBUTIONS

The novelty contributions claimed in this PhD thesis can be summarised as follows:

- Novel *Distributed Failure Detection* method for the detection of failures in the output data streams of network nodes, which can be distributed over several FPGAs.

  The main novelty lies in the fact that the intercommunication of stream processors is done via a switched fabric NoC architecture. Failure detection and isolation mechanisms are embedded into NoC routing switches. Compared to the state of the art, this approach scales much better with the size of the application, can be applied to multi-FPGA systems and allows high-speed communication between the stream processors. It is thus well suited for high performance payload data processing applications.

- Novel *Availability Analysis* method for the prediction of the failure rates and the availability figures for a stream processor in a particular FDIR configuration and radiation environment.

  The most significant novelty is the Block RAM profiling tool. Compared to the standard FPGA toolchain, it allows a much better estimate of susceptible Block RAM bits within a stream processor (or any other circuit) and therefore increases the overall prediction precision for MTBF and availability figures. It advances the state of the art because prior work fully neglected the influence of Block RAM susceptibility. The second novelty is the new fault injection algorithm. The novelty lies in its capability to classify sensitive bits depending on how a system can recover from failures that a triggered by upsets in these sensitive bits. Fault injection systems described in literature are only able to quantify the number of sensitive configuration bits without any further classification. Since the classification enables more general and complex availability models, the state of the art is advanced by the proposed algorithm.

- First systematic survey of the literature on failure detection, isolation and recovery approaches for space-borne SRAM-based FPGAs.

  The survey is on its own a novel contribution to knowledge because no recent work existed, which thoroughly surveyed this rich research field. Together with the included design recommendations, the literature survey can serve as a tutorial for both scientists and engineers who are novices in this field.

9.4 LIMITATIONS AND FUTURE WORK

The Distributed Failure Detection method proved to work reliably throughout the design and validation phase. However, error-free communication depends on the correct choice of timeout values for both the voter module and the multicast mechanism. Since the proof of concept system implements only a small network, the parameters can roughly be estimated. However, in a large network the choice of correct parameters is more critical mainly due to the non-deterministic nature of the NoC architecture. Therefore, it would be interesting to work on analytical and/or simulation models in the future, which could automatically determine the correct timeout values for given network topologies and network traffic conditions.

The Availability Analysis method could be extended by much more complex stochastic Petri net models. Currently, only the availability of a single stream processor is modelled. In the future, the models could be advanced to take also processing chains into account, in which several stream processors are connected in series. In addition, the voter modules are always assumed to be fault-free at present. The fault injection system is designed to not inject faults into configuration memory bits related to IOBs. During experiments, it was accidentally discovered that there might be a chance that SEUs can create shorts in the IOBs, which could potentially damage the device. Thus, it would be valuable to look into this matter in the future as well.

The results of the power, area and performance measurements are limited to the specific proof of concept implementation and thus lack generality. Although they still provide some interesting, qualitative statements, it would be beneficial to repeat the experiments for other types of stream processors in the future. To a certain degree, the same applies to the Availability Analysis case studies and the beam test results. However, the JPEG stream processor was chosen intentionally

as it is believed that it represents a broad range of typical payload data processing algorithms. In any case, the limitation on one kind of stream processor does not impair the validation of the Availability Analysis method, it just makes the resulting figures less general.

BIBLIOGRAPHY

[1] M. Sonza Reorda, L. Sterpone, and M. Violante, "Multiple errors produced by single upsets in FPGA configuration memory: a possible solution," in *European Test Symposium (ETS)*, 2005, pp. 136–141.

[2] R. B. Gardenyes, "Trends and patterns in ASIC and FPGA use in space missions and impact in technology roadmaps of the European Space Agency," Master's thesis, TU Delft, Delft, The Netherlands, 2012.

[3] G. Allen, G. Swift, and C. Carmichael, "Virtex-4QV static SEU characterization summary," NASA Jet Propulsion Laboratory, Xilinx, JPL Publication 08-16 4/08, 2008.

[4] Microsemi, "Radiation-tolerant ProASIC3 low power spaceflight flash FPGAs with Flash*Freeze technology," Microsemi, Datasheet Rev. 5, 2012.

[5] C. Norton, T. Werne, P. Pingree, and S. Geier, "An evaluation of the Xilinx Virtex-4 FPGA for on-board processing in an advanced imaging system," in *IEEE Aerospace conference*, 2009, pp. 1–9.

[6] P. J. Pingree, "Advancing NASA's on-board processing capabilities with reconfigurable FPGA technologies: Opportunities and implications," in *IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW)*, 2010.

[7] F. Siegle, T. Vladimirova, O. Emar, and J. Ilstad, "Mitigation of radiation effects in SRAM-based FPGAs for space applications," in *ACM Computing Surveys*, 2014.

[8] F. Siegle, T. Vladimirova, O. Emar, and J. Ilstad, "Adaptive FDIR strategy for FPGAs hosting partial reconfigurable modules," in *Workshop on Reconfigurable Computing (WRC), HiPEAC Conference*, Jan. 2013.

[9] F. Siegle, T. Vladimirova, O. Emar, and J. Ilstad, "Adaptive FDIR framework for payload data processing systems using reconfig-

urable FPGAs," in *Proc. of 8th NASA/ESA Conference on Adaptive Hardware and Systems*, June 2013.

[10] F. Siegle, T. Vladimirova, O. Emar, and J. Ilstad, "New voter design enabling hot redundancy for asynchronous network nodes," in *Proc. of 9th NASA/ESA Conference on Adaptive Hardware and Systems*, July 2014.

[11] F. Siegle, T. Vladimirova, O. Emar, and J. Ilstad, "Fault detection, isolation and recovery techniques for SRAM-based multi-FPGA systems," in *Proc. of the Military and Aerospace Programmable Logic Devices (MAPLD) Workshop*, May 2014.

[12] F. Siegle, T. Vladimirova, O. Emar, and J. Ilstad, "FDIR techniques for payload streaming applications using SpaceWire-based networks," in *Proc. of the International SpaceWire Conference*, 2014.

[13] F. Siegle, T. Vladimirova, C. Poivey, and O. Emar, "Validation of FDIR strategy for spaceborne SRAM-based FPGAs using proton radiation testing," in *Proc. of the Conference on Radiation Effects on Components and Systems (RADECS 2015)*, 2015.

[14] F. Siegle, T. Vladimirova, O. Emar, and J. Ilstad, "Availability analysis for satellite data processing systems based on SRAM FPGAs," in *IEEE Transactions on Aerospace and Electronic Systems*, 2015.

[15] A. Holmes-Siedle and L. Adams, *Handbook of Radiation Effects*. Oxford University Press, 1993.

[16] ECSS, "Space environment," ESA-ESTEC, Standard ECSS-E-ST-10-04C, 2008.

[17] ECSS, "Methods for the calculation of radiation received and its effects, and a policy for design margins," ESA-ESTEC, Standard ECSS-E-ST-10-12C, 2008.

[18] G. Messenger and M. Ash, *The Effects of Radiation on Electronic Systems*, 2nd ed. New York: Van Nostrand Reinhold, 1992.

[19] C. Dierker, "Fehlertolerante Instrumentenrechner fuer kompakte Kameras auf Raumsonden," Ph.D. dissertation, TU Braunschweig, Braunschweig, Germany, 2007.

[20] Xilinx, "Space-grade Virtex-4QV family overview," Xilinx, Datasheet DS653, 2010.

[21] Xilinx, "Radiation-hardened, space-grade Virtex-5QV family overview," Xilinx, Datasheet DS192, 2012.

[22] P. E. Dodd, M. Shaneyfelt, J. Felix, and J. Schwank, "Production and propagation of single-event transients in high-speed digital logic ICs," *IEEE Transactions on Nuclear Science*, vol. 51, no. 6, pp. 3278–3284, 2004.

[23] G. Swift and G. Allen, "Virtex-5QV static SEU characterization summary," NASA Jet Propulsion Laboratory, Xilinx, Tech. Rep., 2012.

[24] G. Allen, "Virtex-4QV dynamic and mitigated single event upset characterization summary," NASA Jet Propulsion Laboratory, JPL Publication 09-4 01/09, 2009.

[25] H. Quinn, P. Graham, K. Morgan, Z. Baker, M. Caffrey, D. Smith, and R. Bell, "On-orbit results for the Xilinx Virtex-4 FPGA," in *IEEE Radiation Effects Data Workshop (REDW)*, 2012, pp. 1–8.

[26] NASA, "Fault management handbook. Draft 2," National Aeronautics and Space Administration, Handbook NASA-HDBK-1002, 2012.

[27] R. Padovani, "Reconfigurable FPGAs in space - present and future," Presentation at MAPLD Conference, 2005.

[28] C. Carmichael and C. W. Tseng, "Correcting single-event upsets in Virtex-4 FPGA configuration memory," Xilinx, Application Note XAPP1088, 2009.

[29] Xilinx, "Soft error mitigation using prioritized essential bits," Xilinx, Application Note XAPP538, 2012.

[30] J. Heiner, B. Sellers, M. Wirthlin, and J. Kalb, "FPGA partial reconfiguration via configuration scrubbing," in *International Conference on Field Programmable Logic and Applications (FPL)*, 2009, pp. 99–104.

[31] I. Herrera-Alzu and M. Lopez-Vallejo, "Design techniques for Xilinx Virtex FPGA configuration memory scrubbers," *IEEE Transactions on Nuclear Science*, vol. 60, no. 1, pp. 376–385, 2013.

[32] Xilinx, "Virtex-4 FPGA configuration guide," Xilinx, User Guide UG071, 2009.

[33] Xilinx, "Virtex-5 FPGA configuration guide," Xilinx, User Guide UG191, 2012.

[34] M. Lanuzza, P. Zicari, F. Frustaci, S. Perri, and P. Corsonello, "Exploiting self-reconfiguration capability to improve SRAM-based FPGA robustness in space and avionics applications," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 4, no. 1, pp. 8:1–8:22, 2010.

[35] P. M. B. Rao, M. Ebrahimi, R. Seyyedi, and M. B. Tahoori, "Protecting SRAM-based FPGAs against multiple bit upsets using erasure codes," in *51st Annual Design Automation Conference (DAC)*. New York, NY, USA: ACM, 2014, pp. 212:1–212:6.

[36] K. Chapman, "SEU strategies for Virtex-5 devices," Xilinx, Application Note XAPP864, 2010.

[37] A. Sari and M. Psarakis, "Scrubbing-based SEU mitigation approach for systems-on-programmable-chips," in *International Conference on Field-Programmable Technology (FPT)*, 2011, pp. 1–8.

[38] G.-H. Asadi and M. Tahoori, "Soft error mitigation for SRAM-based FPGAs," in *23rd IEEE VLSI Test Symposium (VTS)*, 2005, pp. 207–212.

[39] J. Becker, M. Hubner, G. Hettich, R. Constapel, J. Eisenmann, and J. Luka, "Dynamic and partial FPGA exploitation," *Proceedings of the IEEE*, vol. 95, no. 2, pp. 438–452, 2007.

[40] K. Paulsson, M. Hubner, and J. Becker, "Strategies to on-line failure recovery in self-adaptive systems based on dynamic and partial reconfiguration," in *NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2006, pp. 288–291.

[41] A. Sreeramareddy, J. Josiah, A. Akoglu, and A. Stoica, "SCARS: scalable self-configurable architecture for reusable space systems," in *NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2008, pp. 204–210.

[42] A. Jacobs, G. Cieslewski, A. George, A. Gordon-Ross, and H. Lam, "Reconfigurable fault tolerance: A comprehensive

framework for reliable and adaptive FPGA-based space computing," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 5, no. 4, pp. 21:1–21:30, 2012.

[43] M. Straka, J. Kastil, and Z. Kotasek, "Generic partial dynamic reconfiguration controller for fault tolerant designs based on FPGA," in *The Nordic Microelectronics Event (NORCHIP)*, 2010, pp. 1–4.

[44] J. R. Azambuja, C. Pilotto, and F. Kastensmidt, "Mitigating soft errors in SRAM-based FPGAs by using large grain TMR with selective partial reconfiguration," in *European Conference on Radiation and Its Effects on Components and Systems (RADECS)*, 2008, pp. 288–293.

[45] J. R. Azambuja, F. Sousa, L. Rosa, and F. Kastensmidt, "Evaluating large grain TMR and selective partial reconfiguration for soft error mitigation in SRAM-based FPGAs," in *15th IEEE International On-Line Testing Symposium (IOLTS)*, 2009, pp. 101–106.

[46] X. Iturbe, M. Azkarate, I. Martinez, J. Perez, and A. Astarloa, "A novel SEU, MBU and SHE handling strategy for Xilinx Virtex-4 FPGAs," in *International Conference on Field Programmable Logic and Applications (FPL)*, 2009, pp. 569–573.

[47] G. Nazar, L. Santos, and L. Carro, "Accelerated FPGA repair through shifted scrubbing," in *23rd International Conference on Field Programmable Logic and Applications (FPL)*, 2013, pp. 1–6.

[48] C. Bolchini, A. Miele, and C. Sandionigi, "A novel design methodology for implementing reliability-aware systems on SRAM-based FPGAs," *IEEE Transactions on Computers*, vol. 60, no. 12, pp. 1744–1758, 2011.

[49] M. Berg, C. Poivey, D. Petrick, D. Espinosa, A. Lesea, K. LaBel, M. Friendlich, H. Kim, and A. Phan, "Effectiveness of internal versus external SEU scrubbing mitigation strategies in a Xilinx FPGA: design, test, and analysis," *IEEE Transactions on Nuclear Science*, vol. 55, no. 4, pp. 2259–2266, 2008.

[50] L. Jones, "Single event upset (SEU) detection and correction using Virtex 4 devices," Xilinx, Application Note XAPP714, 2007.

[51] J. Heiner, N. Collins, and M. Wirthlin, "Fault tolerant ICAP controller for high-reliable internal scrubbing," in *IEEE Aerospace Conference*, 2008, pp. 1–10.

[52] A. Ebrahim, K. Benkrid, X. Iturbe, and C. Hong, "A novel high-performance fault-tolerant ICAP controller," in *NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2012, pp. 259–263.

[53] D. McMurtrey, K. S. Morgan, B. Pratt, and M. J. Wirthlin, "Estimating TMR reliability on FPGAs using Markov models," 2008.

[54] B. Bridgford, C. Carmichael, and C. W. Tseng, "Single-event upset mitigation selection guide," Xilinx, Application Note XAPP987, 2008.

[55] P. Adell and G. Allen, "Assessing and mitigating radiation effects in Xilinx FPGAs," NASA Jet Propulsion Laboratory, JPL Publication 08-9 2/08, 2008.

[56] Xilinx, "TMRTool," [Online] http://www.xilinx.com/ise/optional_prod/tmrtool.htm.

[57] Mentor Graphics, "Precision Hi-Rel technology overview," [Online] http://www.mentor.com/products/fpga/.

[58] Synopsis, "Synplify Premier. fast, reliable FPGA implementation and debug," [Online] http://www.synopsys.com/Tools/Implementation/FPGAImplementation/CapsuleModule/syn_prem_ds.pdf.

[59] Brigham Young University, "BYU EDIF tools home page," [Online] http://reliability.ee.byu.edu/edif/.

[60] H. Quinn, K. Morgan, P. Graham, J. Krone, M. Caffrey, and K. Lundgreen, "Domain crossing errors: Limitations on single device triple-modular redundancy circuits in Xilinx FPGAs," *IEEE Transactions on Nuclear Science*, vol. 54, no. 6, pp. 2037–2043, 2007.

[61] B. Pratt, M. Caffrey, P. Graham, K. Morgan, and M. Wirthlin, "Improving FPGA design robustness with partial TMR," in *44th IEEE International Reliability Physics Symposium (IRPS)*, 2006, pp. 226–232.

[62] B. Pratt, M. Caffrey, J. Carroll, P. Graham, K. Morgan, and M. Wirthlin, "Fine-grain SEU mitigation for FPGAs using partial TMR," *IEEE Transactions on Nuclear Science*, vol. 55, no. 4, pp. 2274–2280, 2008.

[63] K. Morgan, M. Caffrey, P. Graham, E. Johnson, B. Pratt, and M. Wirthlin, "SEU-induced persistent error propagation in FPGAs," *IEEE Transactions on Nuclear Science*, vol. 52, no. 6, pp. 2438–2445, 2005.

[64] F. L. Kastensmidt, L. Sterpone, L. Carro, and M. Reorda, "On the optimal design of triple modular redundancy logic for SRAM-based FPGAs," in *Design, Automation and Test in Europe (DATE)*, vol. 2, 2005, pp. 1290–1295.

[65] J. Johnson, W. Howes, M. Wirthlin, D. McMurtrey, M. Caffrey, P. Graham, and K. Morgan, "Using duplication with compare for on-line error detection in FPGA-based designs," in *IEEE Aerospace Conference*, 2008, pp. 1–11.

[66] J.-P. Anderson, B. Nelson, and M. Wirthlin, "Using statistical models with duplication and compare for reduced cost FPGA reliability," in *IEEE Aerospace Conference*, 2010, pp. 1–8.

[67] S. Habinc, "Suitability of reprogrammable FPGAs in space applications," Gaisler Research, Feasibility Report, 2002.

[68] M. Straka, J. Kastil, and Z. Kotasek, "Fault tolerant structure for SRAM-based FPGA via partial dynamic reconfiguration," in *13th Euromicro Conference on Digital System Design: Architectures, Methods and Tools (DSD)*, 2010, pp. 365–372.

[69] J. Snodgrass, "Low-power fault tolerance for spacecraft FPGA-based numerical computing." Ph.D. dissertation, Naval Postgraduate School, Monterey, CA, U.S.A., 2006.

[70] M. Sullivan, "Reduced precision redundancy applied to arithmetic operations in field programmable gate arrays for satellite control and sensor systems," Master's thesis, Naval Postgraduate School, Monterey, California, 2008.

[71] M. A. Sullivan, H. Loomis, and A. Ross, "Employment of reduced precision redundancy for fault tolerant FPGA applications," in *17th IEEE Symposium on Field Programmable Custom Computing Machines (FCCM)*, 2009, pp. 283–286.

[72] A. Gavros, H. Loomis, and A. Ross, "Reduced precision redundancy in a Radix-4 FFT implementation on a Field Programmable Gate Array," in *IEEE Aerospace Conference*, 2011, pp. 1–15.

[73] B. Pratt, M. Fuller, and M. Wirthlin, "Reduced-precision redundancy on FPGAs," *International Journal of Reconfigurable Computing*, vol. 2011, 2011.

[74] B. Pratt, M. Fuller, M. Rice, and M. Wirthlin, "Reduced-precision redundancy for reliable FPGA communications systems in high-radiation environments," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 49, no. 1, pp. 369–380, 2013.

[75] G. Burke and S. Taft, "Fault tolerant state machines," in *Military and Aerospace Programmable Logic Devices Workshop (MAPLD)*. Jet Propulsion Laboratory, 2004.

[76] K. Morgan, D. McMurtrey, B. Pratt, and M. Wirthlin, "A comparison of TMR with alternative fault-tolerant design techniques for FPGAs," *IEEE Transactions on Nuclear Science*, vol. 54, no. 6, pp. 2065–2072, 2007.

[77] I. Skliarova, "Self-correction of FPGA-based control units," in *2nd International Conference on Embedded Software and Systems (ICESS)*. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 310–319.

[78] K.-H. Huang and J. Abraham, "Algorithm-based fault tolerance for matrix operations," *IEEE Transactions on Computers*, vol. C-33, no. 6, pp. 518–528, 1984.

[79] A. Jacobs, G. Cieslewski, and A. George, "Overhead and reliability analysis of algorithm-based fault tolerance in FPGA systems," in *22nd International Conference on Field Programmable Logic and Applications (FPL)*, 2012, pp. 300–306.

[80] G. Miller, C. Carmichael, and G. Swift, "Single-event upset mitigation for Xilinx FPGA block memories," Xilinx, Application Note XAPP962, 2008.

[81] N. Rollins, M. Fuller, and M. Wirthlin, "A comparison of fault-tolerant memories in SRAM-based FPGAs," in *IEEE Aerospace Conference*, 2010, pp. 1–12.

[82] M. Sonza Reorda, L. Sterpone, and M. Violante, "Efficient estimation of SEU effects in SRAM-based FPGAs," in *11th IEEE International On-Line Testing Symposium (IOLTS)*, 2005, pp. 54–59.

[83] L. Sterpone, M. Reorda, and M. Violante, "RoRA: a reliability-oriented place and route algorithm for SRAM-based FPGAs," in *Research in Microelectronics and Electronics, 2005 PhD*, vol. 1, 2005, pp. 173–176.

[84] L. Sterpone and M. Violante, "A new reliability-oriented place and route algorithm for SRAM-based FPGAs," *IEEE Transactions on Computers*, vol. 55, no. 6, pp. 732 –744, 2006.

[85] L. Sterpone and N. Battezzati, "A novel design flow for the performance optimization of fault tolerant circuits on SRAM-based FPGA's," in *NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2008, pp. 157–163.

[86] L. Sterpone and M. Violante, "Static and dynamic analysis of SEU effects in SRAM-based FPGAs," in *12th IEEE European Test Symposium (ETS)*, 2007, pp. 159–164.

[87] L. Sterpone and M. Violante, "A new algorithm for the analysis of the MCUs sensitiveness of TMR architectures in SRAM-Based FPGAs," *IEEE Transactions on Nuclear Science*, vol. 55, no. 4, pp. 2019–2027, 2008.

[88] L. Sterpone and N. Battezzati, "A new placement algorithm for the mitigation of multiple cell upsets in SRAM-based FPGAs," in *Design, Automation Test in Europe (DATE)*, 2010, pp. 1231–1236.

[89] C. Lavin, M. Padilla, J. Lamprecht, P. Lundrigan, B. Nelson, and B. Hutchings, "RapidSmith: Do-it-yourself CAD tools for Xilinx FPGAs," in *21th International Workshop on Field-Programmable Logic and Applications (FPL)*, 2011.

[90] A. Sari, D. Agiakatsikas, and M. Psarakis, "A soft error vulnerability analysis framework for Xilinx FPGAs," in *ACM/SIGDA International Symposium on Field-programmable Gate Arrays (FPGA)*. New York, NY, USA: ACM, 2014, pp. 237–240.

[91] J. L. Barth, K. A. LaBel, and C. Poivey, "Radiation assurance for the space environment," in *International Conference on Integrated Circuit Design and Technology (ICICDT)*, 2004, pp. 323–333.

[92] ESA/ESCIES, "Radiation test facilities," [Online] https://escies.org/webdocument/showArticle?id=230&groupid=6.

[93] H. Quinn, P. Graham, J. Krone, M. Caffrey, and S. Rezgui, "Radiation-induced multi-bit upsets in SRAM-based FPGAs," *IEEE Transactions on Nuclear Science*, vol. 52, no. 6, pp. 2455–2461, 2005.

[94] H. Quinn, K. Morgan, P. Graham, J. Krone, and M. Caffrey, "A review of Xilinx FPGA architectural reliability concerns from Virtex to Virtex-5," in *9th European Conference on Radiation and Its Effects on Components and Systems (RADECS)*, 2007, pp. 1–8.

[95] H. Quinn, K. Morgan, P. Graham, J. Krone, and M. Caffrey, "Static proton and heavy ion testing of the Xilinx Virtex-5 device," in *IEEE Radiation Effects Data Workshop (REDW)*, 2007, pp. 177–184.

[96] H. Quinn, G. Allen, G. Swift, C. W. Tseng, P. Graham, K. Morgan, and P. Ostler, "SEU-susceptibility of logical constants in Xilinx FPGA designs," *IEEE Transactions on Nuclear Science*, vol. 56, no. 6, pp. 3527–3533, 2009.

[97] H. Quinn, P. Graham, M. Wirthlin, B. Pratt, K. Morgan, M. Caffrey, and J. Krone, "A test methodology for determining space readiness of Xilinx SRAM-based FPGA devices and designs," *IEEE Transactions on Instrumentation and Measurement*, vol. 58, no. 10, pp. 3380–3395, 2009.

[98] G. Allen, G. Swift, and G. Miller, "Upset characterization and test methodology of the PowerPC405 hard-core processor embedded in Xilinx Field Programmable Gate Arrays," in *IEEE Radiation Effects Data Workshop (REDW)*, 2007, pp. 167–171.

[99] G. Swift, G. Allen, C. W. Tseng, C. Carmichael, G. Miller, and J. George, "Static upset characteristics of the 90nm Virtex-4QV FPGAs," in *IEEE Radiation Effects Data Workshop (REDW)*, 2008, pp. 98–105.

[100] M. Alderighi, F. Casini, S. D'Angelo, M. Mancini, S. Pastore, and G. Sechi, "Evaluation of single event upset mitigation schemes for SRAM based FPGAs using the FLIPPER fault injection platform," in *22nd IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT)*, 2007, pp. 105–113.

[101] M. Alderighi, F. Casini, S. D'Angelo, M. Mancini, D. Codinachs, S. Pastore, C. Poivey, G. Sechi, G. Sorrenti, and R. Weigand,

"Experimental validation of fault injection analyses by the FLIPPER tool," *IEEE Transactions on Nuclear Science*, vol. 57, no. 4, pp. 2129–2134, 2010.

[102] M. Alderighi, F. Casini, S. D'Angelo, M. Mancini, D. Codinachs, S. Pastore, G. Sorrenti, L. Sterpone, R. Weigand, and M. Violante, "Robustness analysis of soft error accumulation in SRAM-FPGAs using FLIPPER and STAR/RoRA," in *European Conference on Radiation and Its Effects on Components and Systems (RADECS)*, 2008, pp. 157–161.

[103] M. A. Aguirre, V. Baena, J. Tombs, and M. Violante, "A new approach to estimate the effect of single event transients in complex circuits," *IEEE Transactions on Nuclear Science*, vol. 54, no. 4, pp. 1018–1024, 2007.

[104] M. A. Aguirre, J. N. Tombs, F. Muoz, V. Baena, H. Guzman, J. Napoles, A. Torralba, A. Fernandez-Leon, F. Tortosa-Lopez, and D. Merodio, "Selective protection analysis using a SEU emulator: Testing protocol and case study over the Leon2 processor," *IEEE Transactions on Nuclear Science*, vol. 54, no. 4, pp. 951–956, 2007.

[105] J. Napoles, H. Guzman-Miranda, M. Aguirre, J. Tombs, J. Mogollon, R. Palomo, and A. Vega-Leal, "A complete emulation system for single event effects analysis," in *4th Southern Conference on Programmable Logic (SPL)*, 2008, pp. 213–216.

[106] H. Guzman-Miranda, M. Aguirre, and J. Tombs, "A non-invasive system for the measurement of the robustness of microprocessor-type architectures against radiation-induced soft errors," in *IEEE Instrumentation and Measurement Technology Conference Proceedings (IMTC)*, 2008, pp. 2009–2014.

[107] J. M. Mogollon, H. Guzman-Miranda, J. Napoles, J. Barrientos, and M. Aguirre, "FTUNSHADES2: a novel platform for early evaluation of robustness against SEE," in *12th European Conference on Radiation and Its Effects on Components and Systems (RADECS)*, 2011, pp. 169–174.

[108] P. Ostler, M. Caffrey, D. Gibelyou, P. Graham, K. Morgan, B. Pratt, H. Quinn, and M. Wirthlin, "SRAM FPGA reliability analysis for harsh radiation environments," *IEEE Transactions on Nuclear Science*, vol. 56, no. 6, pp. 3519–3526, 2009.

[109] J. Kastil, M. Straka, L. Miculka, and Z. Kotasek, "Dependability analysis of fault tolerant systems based on partial dynamic reconfiguration implemented into FPGA," in *15th Euromicro Conference on Digital System Design (DSD)*, 2012, pp. 250–257.

[110] Q. Martin and A. George, "Scrubbing optimization via availability prediction (SOAP) for reconfigurable space computing," in *IEEE Conference on High Performance Extreme Computing (HPEC)*, 2012, pp. 1–6.

[111] K. Hoque, O. Mohamed, Y. Savaria, and C. Thibeault, "Early analysis of soft error effects for aerospace applications using probabilistic model checking," in *Formal Techniques for Safety-Critical Systems*. Springer International Publishing, 2014, vol. 419, pp. 54–70.

[112] K. Hoque, O. Mohamed, Y. Savaria, and C. Thibeault, "Probabilistic model checking based DAL analysis to optimize a combined TMR-blind-scrubbing mitigation technique for FPGA-based aerospace applications," in *Twelfth ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE)*, Oct 2014, pp. 175–184.

[113] J. Hagemeyer, A. Hilgenstein, D. Jungewelter, D. Cozzi, C. Felicetti, U. Rueckert, S. Korf, M. Koester, F. Margaglia, M. Porrmann, F. Dittmann, M. Ditze, J. Harris, L. Sterpone, and J. Ilstad, "A scalable platform for run-time reconfigurable satellite payload processing," in *NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2012, pp. 9–16.

[114] M. Koester, W. Luk, J. Hagemeyer, M. Porrmann, and U. Ruckert, "Design optimizations for tiled partially reconfigurable systems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 6, pp. 1048–1061, 2011.

[115] L. Sterpone, F. Margaglia, M. Koester, J. Hagemeyer, and M. Porrmann, "Analysis of SEU effects in partially reconfigurable SoPCs," in *NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2011, pp. 129 –136.

[116] X. Iturbe, K. Benkrid, A. Erdogan, T. Arslan, M. Azkarate, I. Martinez, and A. Perez, "R3TOS: A reliable reconfigurable real-time operating system," in *NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2010, pp. 99–104.

[117] X. Iturbe, K. Benkrid, T. Arslan, I. Martinez, and M. Azkarate, "ATB: area-time response balancing algorithm for scheduling real-time hardware tasks," in *International Conference on Field-Programmable Technology (FPT)*, 2010, pp. 224–232.

[118] C. Hong, K. Benkrid, X. Iturbe, A. Erdogan, and T. Arslan, "An FPGA task allocator with preliminary first-fit 2D packing algorithms," in *NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2011, pp. 264–270.

[119] X. Iturbe, K. Benkrid, T. Arslan, R. Torrego, and I. Martinez, "Methods and mechanisms for hardware multitasking: Executing and synchronizing fully relocatable hardware tasks in Xilinx FPGAs," in *International Conference on Field Programmable Logic and Applications (FPL)*, 2011, pp. 295–300.

[120] X. Iturbe, K. Benkrid, A. Ebrahim, C. Hong, T. Arslan, and I. Martinez, "Snake: An efficient strategy for the reuse of circuitry and partial computation results in high-performance reconfigurable computing," in *International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, 2011, pp. 182–189.

[121] S. Yousuf, A. Jacobs, and A. Gordon-Ross, "Partially reconfigurable system-on-chips for adaptive fault tolerance," in *International Conference on Field Programmable Logic and Applications (FPL)*, 2011, pp. 1–8.

[122] G. Cieslewski, A. D. George, and A. Jacobs, "Acceleration of FPGA fault injection through multi-bit testing," in *International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, 2010, pp. 218–224.

[123] H. Michel, F. Bubenhagen, B. Fiethe, H. Michalik, B. Osterloh, W. Sullivan, A. Wishart, J. Ilstad, and S. Habinc, "AMBA to SoCWire network on chip bridge as a backbone for a dynamic reconfigurable processing unit," in *NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2011, pp. 227–233.

[124] B. Osterloh, H. Michalik, B. Fiethe, and F. Bubenhagen, "Enhancements of reconfigurable system-on-chip data processing units for space application," in *NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2007, pp. 258–262.

[125] B. Osterloh, H. Michalik, B. Fiethe, and K. Kotarowski, "SoCWire: a network-on-chip approach for reconfigurable

system-on-chip designs in space applications," in *NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2008, pp. 51–56.

[126] ECSS, "Space engineering: SpaceWire - links, nodes, routers and networks. ECSS-E-50-12A," ESA/ESTEC, Tech. Rep., 2003.

[127] B. Osterloh, H. Michalik, S. Habinc, and B. Fiethe, "Dynamic partial reconfiguration in space applications," in *NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2009, pp. 336–343.

[128] B. Osterloh, H. Michalik, B. Fiethe, and F. Bubenhagen, "Architecture verification of the SoCWire NoC approach for safe dynamic partial reconfiguration in space applications," in *NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2010, pp. 1–8.

[129] H. Michel, A. Belger, F. Bubenhagen, B. Fiethe, H. Michalik, W. Sullivan, A. Wishart, and J. Ilstad, "The SoCWire protocol (SoCP): a flexible and minimal protocol for a network-on-chip," in *NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2012, pp. 1–8.

[130] M. Straka, J. Tobola, and Z. Kotasek, "Checker design for online testing of Xilinx FPGA communication protocols," in *22nd IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT)*, 2007, pp. 152–160.

[131] M. Straka, L. Miculka, J. Kastil, and Z. Kotasek, "Test platform for fault tolerant systems design properties verification," in *IEEE 15th International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, 2012, pp. 336–341.

[132] S. Mitra and E. J. Mccluskey, "Word-voter: A new voter design for triple modular redundant systems," in *Proc. VLSI Test symposium*, 2000, pp. 465–470.

[133] ECSS, "Space engineering: Space environment. ECSS-Q-ST-30-09C," ESA/ESTEC, Tech. Rep., 2008.

[134] ECSS, "Space engineering: Space environment. ECSS-E-ST-10-04C," ESA/ESTEC, Tech. Rep., 2008.

[135] A. Agresti and B. A. Coull, "Approximate is better than 'exact' for interval estimation of binomial proportions," *The American Statistician*, vol. 52, no. 2, pp. 119–126, 1998.

[136] Mentor Graphics, "ModelSim ASIC and FPGA design," [Online] http://www.mentor.com/products/fv/modelsim.

[137] J. Pellish, M. Xapsos, C. Stauffer, T. Jordan, A. Sanders, R. Ladbury, T. Oldham, P. Marshall, D. Heidel, and K. Rodbell, "Impact of spacecraft shielding on direct ionization soft error rates for sub-130 nm technologies," *IEEE Transactions on Nuclear Science*, vol. 57, no. 6, pp. 3183–3189, Dec 2010.

[138] A. Zimmermann, "Modeling and evaluation of stochastic petri nets with TimeNET 4.1," in *6th International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS)*, 2012, pp. 54–63.

[139] Microsemi, "ProASIC3E flash family FPGAs. Datasheet DS0098, revision 15," Microsemi, Tech. Rep., 2015.

[140] Aeroflex Gaisler, "SPWRTC development unit. User manual revision 1.1," Aeroflex Gaisler AB, Tech. Rep., 2008.

[141] Michal Krepa, "JPEG encoder," [Online] www.opencores.org/project,mkjpeg.

[142] "Packet telemetry. CCSDS 102.0-B-5," Consultative Committee for Space Data Systems, Tech. Rep., 2000.

[143] Xilinx, "Partial reconfiguration user guide," Xilinx, User Guide UG702, 2012.

[144] STAR-Dundee, "SpaceWire IP cores," [Online] www.star-dundee.com/products/spacewire-ip-cores.

[145] Cobham Gaisler, "GRLIB IP library user's manual, version 1.4.1," Cobham Gaisler, Tech. Rep., 2015.

[146] The RTEMS Project, "RTEMS real time operating system (RTOS)," [Online] www.rtems.org.

[147] "Qt open source version," [Online] http://www.qt.io/download-open-source.

[148] "SQLite home page," [Online] http://www.sqlite.org.

[149] "OpenCV home page," [Online] http://www.opencv.org.

[150] "IEEE standard for verilog hardware description language," *IEEE Std 1364-2005 (Revision of IEEE Std 1364-2001)*, 2006.

[151] "Logic design: TMRTool," [Online] http://www.xilinx.com/ise/optional_prod/tmrtool.htm.

[152] TRAD, "OMERE software," [Online] http://www.trad.fr/OMERE-Software.html.