B.Sc.Eng. Thesis
Bachelor of Science in Electronic Engineering

DTU Space
National Space Institute

Fault Tolerance Techniques for FPGA-based Space Applications

A study of the effects of radiation in SRAM-based FPGAs and their solutions

Gabriel Cobos Tello

DTU Space 2019

DTU Space
National Space Institute
Technical University of Denmark

Elektrovej
Building 327
2800 Kongens Lyngby, Denmark
Phone +45 4525 95 00
office@space.dtu.dk
www.space.dtu.dk/English

Summary

This thesis focuses on mitigation techniques for SRAM-based FPGAs and how they improve design reliability. The FPGA implemented design is a basic AVR CPU running a matrix multiplication workload. The process followed starts with faults being injected into the basic design to obtain a reliability baseline. Mitigation techniques are then used, in the form of basic TMR and hamming code EDAC, to improve the reliability of the design. Then an iterative process of mitigation techniques and fault injection is carried out to optimise resources and maximise design reliability. This is all done in hopes of presenting a realistic approximation of space system design and the margins this area of engineering is limited by.

The mitigation techniques where all developed for the purpose of the thesis, using state of the art techniques as reference. The CPU was obtained from OpenCores alongside with the tool-chain required to program the custom workload. The fault injection methods where provided by David de Andrés and Ilya Tuvoz, from UPV's department of computer engineering.

Preface

A symphony of Space, Radiation and Electronics

Space has always been a mysterious field for science and technology, so naturally, I felt tremendously attracted to it. This attraction consisted mostly of the kind of infantile curiosity that characterises any endeavor with lots of promise and an unsuspected difficulty. The reasons I use this two words to describe what I experienced while working in this thesis, is because through the course of the project there was a recurring theme of fascination, followed by frustration at how complicated bringing the theory to practice was. This was something I overlooked when I first proposed a broad idea for a Thesis to my supervisor David de Andrés. At the time I just wanted to work with FPGAs and if possible get a head start at what I hoped one day would become a successful space career. David was the one to propose the topic of fault tolerance in FPGAs, offering his supervision and mentoring. I promptly accepted and started looking for supervisors at DTU Space, since I was certain I wanted to carry out the project in a department specialized in space research. Eventually Jose M.G Merayo was the one to show interest in the nature of my thesis and he offered to become the DTU supervisor. With a full team of supervisors, also including Carlos Dominguez, a computer science professor at UPV, I started work on this thesis.

DTU Space, June 20, 2019

Gabriel Cobos Tello

Acknowledgements

I would like to thank Carlos Dominguez for his help as the administrative supervisor of this thesis. It was due to his efforts that it was possible for me to fulfil the complex task of writing a thesis in a foreign university.

Of course any effort would have been in vain without the help and involvement of Jose M.G Merayo, my supervisor at DTU. During the first months our weekly meetings allowed me to grasp the basics of the project while also bringing a sense of security to the entire process. It would be hard to imagine how alienating it would have been to work on a thesis at a new university without the constant care and support of Jose.

David de Andres was my secondary supervisor at UPV, yet his involvement was total. It is thanks to his generosity and patience during the first times we met that the thesis happened in the first place. I can not begin to imagine the amount of hours invested by him from Valencia to assist me in my work here at DTU. Not only did he provide the methods for me to test my designs, but he also made sure they worked, alongside Ilya Tuvoz his PhD student and co-developer of the fault injection mechanisms. In my opinion David proved to be someone truly invested in teaching others, as he would always answer any queries without the slightest trace of annoyance, always looking out for the student's best interests. I can honestly confess that without his help I would have never been able to break the barrier of entry for this topic, and for that I am truly grateful.

To my parents I also owe a well deserved thank you, for it is through their sacrifices that I can focus on my studies and experience adventures like studying abroad. Specially my mother, for whom I have no words that can describe the extend to which she has gone to make sure I grow and flourish. If I ever achieve anything it is thanks to you, for you gave me the base upon which to build my life. Yet, you know while reading this, that these are just words and what truly counts are the actions taken every day. The gestures that allow us to rely on each other and develop as people. This is something I don't share with anyone else and for that we will always be special for each other.

A Papa: Si no fuera por lo mucho que trabajas y por como siempre nos tienes en mente a mi y a mama, jamas podría hacer las cosas que hago. Es gracias a ti que algún día pueda llegar a ser alguien que contribuya a la sociedad, pudiendo centrarme hoy por hoy plenamente en mis estudios, pero eso ya lo sabes. Lo que quizás no sepas es que incluso si algún día decides que tienes que descansar, por poco que sea,

nada cambiara. Yo siempre te voy a querer igual, es algo crónico e incondicional, no depende ni del trabajo ni de tus acciones. Yo siempre voy a conseguir llegar a cumplir mis objetivos, porque ese es el tipo de persona que me as ayudado a ser. Yo siempre estaré listo para cambiar de rol cuando lo necesites, porque a veces es necesario que la persona que mas se apoya en ti se gire, y por una vez de de su hombro a apoyar, para con suerte alibiar la carga generada a lo largo de los años. Con amor, Gabriel.

Contents

List of Figures

List of Acronyms

ALU	Arithmetic Logic Unit
ASIC	Application Specific Integrated Circuits
ASMBL	Advanced Silicon Modular Block
BRAM	Block RAMs
CB	Critical Bits
CPU	Central Processing Unit
CLB	Configurable Logic Blocks
CMT	Clock Management Tiles
COTS	Commercial of the Shelf
DUT	Design Under Test
EB	Essential Bits
EDA	Electronic Design Automation
ESA	European Space Agency
EDC	Error Detection Code
ECC	Error Correction Code
EDAC	Error Detection and Correction Code
FF	Flip Flops
FIT	Failure in Time
FPGA	Field Programmable Gate Array
GCR	Galactic Cosmic Radiation
HDL	Hardware Description Language

IP	Internet Protocol
IR	Interconnect Resources
INO	IN and Output Block
IOB	Input and Output Blocks
LUT	Look Up Table
MAJ	Majority Voter
MUX	Multiplexers
MOSFET	Metal Oxide Semiconductor Field-Effect Transistors
NASA	National Aeronautics and Space Administration
PC	Program Counter
PL	Programmable Logic
PS	Programmable System
PoF	Probability of Failure
PIP	Programmable Interconnect Points
RoF	Rate of Faults
SM	Switch Matrices
SOI	Sillicon on Insulator
SEU	Single Event Upsets
SET	Single Event Transient
SOC	System on a Chip
SAA	South Atlantic Anomaly
SRAM	Static Random Access Memory
TMR	Triple Modular Redundancy
TID	Total Ionising Dose
UART	Universal Asynchronous Receiver-Transmitter
XOR	Exclusive OR

CHAPTER 1

Introduction

Several developments through out the years have placed FPGAs in an interesting spot in relation to space applications. One such development, shared by most electronic devices, is the trend of constant increasing density of logic resources. This trend, paired with the significant improvements in the clock speeds at which this devices can operate, has allowed for higher performance and lower power consumption in FPGAs when compared to microprocessors or application specific integrated circuits (ASICs) [11]. These events see FPGAs being used in the place of ASICs in many systems. This being the case in space applications as well, where FPGA's re-programmability would in theory be of most use. Either for long space or short near earth missions, this re-programmability grants FPGAs and the systems within which they operate, great flexibility in the form of system updates that can be implemented at a hardware level, or the ability of fixing design issues once the mission has already started.

Of course all these advantages come at a cost. The reason why FPGAs stand in an interesting spot in relation to space applications is because their biggest strengths are also their biggest weakness. Both their great logic density and field programmability make FPGAs extremely sensible to radiation effects, specially in radiation rich environments like space. FPGAs that try to amend this issue exist and are known as Space-Grade FPGAs but they also come with some disadvantages. Normal FPGAs, known as commercial of the shelf (COTS) FPGAs, can have up to 2.5 times the logic resources of a Space-Grade, faster switch time, less power consumption and the support of modern design tools [5]. This leaves system designers at a cross roads, in which they have to decide whether to prioritize reliability at the cost of higher performance rates, or prioritize more modern COTS devices and invest in expensive testing to apply fault tolerant technology.

The topic hereby presented will assume the latter option is chosen and will be a detailed analysis of fault tolerance techniques for SRAM-based FPGAs employed in space applications.

CHAPTER 2

Overview on Space Radiation in Electronics

This overview will focus on the theory related to the general space radiation environment and the effect of radiation on electronics as a whole. Yet, types of radiation do not change throughout the entire region of space in our known solar system, with only the quantities and spectrum of these varying depending on the location being studied [9]. Hence, the thesis will be focused on the near-Earth radiation environment, since it provides a milder radiation environment and good sources of data from ESA and NASA, without limiting the types of radiation to be discussed. It is important to understand the general concepts behind the topic of space radiation on electronics before progressing on to the focus of the thesis, as this very understanding is the justification for the subjects explained in detail in chapter 3.

2.1 Near-Earth Radiation Environment

The near-Earth radiation environment can be divided into a trapped radiation environment and a transient radiation environment [9], with some secondary sources of radiation also appearing occasionally. The trapped radiation environment consists of charged particles that are confined to certain regions due to Earth's magnetic field. These regions are denominated Van Allen Belts and normally there are one proton and two electron belts, although this can change due to strong solar events. The transient radiation environment is composed of solar wind, solar flares and Galactic Cosmic Radiation (GCR), which are not only present in the near-Earth region but also in the interplanetary space regions.

Trapped Radiation Environments The trapped radiation belts extend from approximately 500 km to about 12 Earth radii (roughly 76,000 km) [9]. It is within this range that the previously mentioned electron and proton bands are located. These

bands are populated with electrons, protons and small amounts of low energy heavy ions that are trapped by the Earth's magnetic field, above the dense atmosphere. These particles gyrate around, bounce along magnetic field lines and are reflected back and forth between pairs of regions with maximum magnetic field strength along their trajectory through opposite hemispheres[15].

Given how trapped radiation is so heavily dependent on the Earth's magnetic field lines, there is a special phenomenon that has to be taken into account, the South Atlantic Anomaly (SAA). The Earth's magnetic field axis does not point to the geographic north and does not pass directly through the center of the Earth, this causes a deformation in the magnetic field over the South Atlantic and over Southeast Asia, with the net effect of an increase in radiation harshness from trapped particles in this regions [9]. The effects of the SAA are so severe that they are always factored into the calculations of the orbits of satellites.

Transient Radiation Environments Solar wind is composed of low energy electrons and protons and its considered to be negligible when compared to other sources of radiation, with the exception of externally mounted spacecraft components where it can sometimes be energetically significant. Solar flares are the other source of radiation originating from our sun, in particular when a magnetic disruption in the solar photosphere occurs, causing a variety of radiation types and energies to erupt [9]. These eruptions can produce energetic protons and heavy ions that when impacting on electronic devices will cause unwanted effects. The last of the three transient radiation sources are GCRs, these are composed of electrons, protons and heavy ions, all of which are believed to have an origin outside the solar system and to be omnidirectional [9]. Just like solar flares GCRs can cause unwanted effects in electronic components.

Solar flares and GCRs always have to be taken into consideration alongside the Earth's magnetic field. The magnetic forces originating from the Earth also act on the charged particles contained in these sources of radiation, meaning that for a given magnetic field strength only a certain number of sufficiently energetic particles will penetrate without being reflected along the magnetic field lines.

The radiation sources described will, in general terms, produce ionizing particles that might impact electronic system causing unwanted effects. The amount and frequency by which this particles may impact has to be estimated based on the scenario where the electronic system will be deployed. It is in this estimation where an advanced understanding of space physics and weather is required. Without good approximations to the radiation harshness of the scenario, the system designers might implement either too much or too little robustness into their system. Thus causing problems in the long run, such as production delays or even system failures during real missions.

On top of a good understanding of the scenario, knowledge on the general effects of ionizing radiation in electronics (semiconductor devices) is required in order to

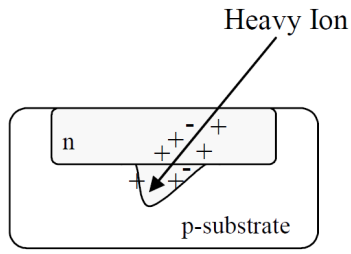

Figure 2.1: Ionizing particle colliding with silicon semiconductor surface [10].

successfully predict the behaviour of radiation on any specific system.

2.2 Semiconductors: Single Event Effects

Just like the radiation environment is narrowed down to that of the low-Earth orbit, the effects of radiation on electronic devices will also be narrowed down to Single event effects, without taking into consideration cumulative effects. The reason being that the techniques deployed in later chapters only protect against the effects of particle impacts, also known as Single Event Upsets (SEU), and not the Total Ionising Dose (TID) responsible for some of the possible permanent errors within the device.

When studying the effects of ionizing particles colliding with electronic devices its important to focus on their most basic component, the transistor. In specific Metal Oxide Semiconductor Field-Effect Transistors (MOSFET), which have been dominant in the fabrication of integrated digital circuits. When an ionizing particle impacts the semiconductor region of the transistor it loses energy via the production of free electron-hole pairs in its trajectory [10], figure 2.1. The number of electron-hole pairs is proportional to the deposited energy by the particle and thus the effects are proportional to the particle energy too. Depending on the orientation of the electrons or holes produced, the electrical current in the transistor might start flowing or be stopped. This might cause the originally intended behaviour of the transistor to change, in turn affecting the electronic device's behaviour at a digital level too. This phenomenon is referred to as a SEU.

There are two types of SEU, destructive or non-destructive, based on the damage they cause to the element at impact. Destructive SEU will lead to hard errors that will permanently affect the element until it is repaired or replaced. Non-destructive SEU will lead to transient faults that might ripple through the device where the element resides, this is known as a soft error. Non-destructive SEU are more common and depending on the device their effects will vary greatly. For instance SEUs in FPGAs have very particular effects that make this type of devices very interesting to study, hence the focus of the thesis as presented in the next chapter.

CHAPTER 3

FPGAs in Space Applications

As mentioned in the introduction, there is a general interest within the space industry to see COTS FPGAs achieve high levels of reliability. This is the core principle behind the focus of the thesis and the reason why the device of choice throughout the project will be an SRAM-based FPGA, specifically the zybo-z7020. This kind of FPGAs grant performance, flexibility and scheduling benefits to system designers, when compared to other more robust FPGAs such as the fuse or flash based FPGAs [6]. Yet, like all static memory cells, SRAM-based configuration cells used in SRAM FPGAs are susceptible to data corruption from high-energy radiation [19]. Furthermore, given how the number of configuration cells in FPGAs are increasing due to logic resources density, as previously mentioned, upsets within the configuration memory are also increasing. This kind of upsets, when occurring in re-configurable systems like FPGAs, are especially troublesome since these cells specify the operation of the re-configurable fabric and thus the architecture of the digital circuit designed. Based on this concepts, a variety of fault tolerance techniques will be discussed in section 3.1, with special consideration been taken for the effects of SEU in FPGAs as described in detail in section 3.2.

3.1 Dependability and Security in the Face of SEU

State-of-the-Art Technological advancements in fabrication technologies, following Moore's law, are progressively making electronic devices more sensible to external disturbances [3]. From these, ionizing radiation is of special interest to us. Normally and for many years, shielding was the main prevention security measure for spacecraft systems, as this solution was enough to avoid errors from radiation effects [10]. Nowadays, due to the aforementioned technological advancements, shielding is no longer sufficient as a single means of dependability. This is why mitigation and removal techniques are being developed in order to avoid faults in the electronic devices aboard spacecrafts. A classification methodology from [2] is used in order to organise these state-of-the-art techniques:

1. Fault prevention, such as:
    a) Epitaxial CMOS processes
    b) Advanced process such as Sillicon on Insulator (SOI).
    c) Shielding
2. Fault Tolerance, such as:
    a) Detection techniques:
        i. Hardware redundancy
        ii. Time redundancy
        iii. Error Detection code (EDC)
        iv. Self-checker techniques
    a) Mitigation techniques:
        i. Triple Modular Redundancy (TMR)
        ii. Multiple redundancy with voting
        iii. Error Detection and Correction Code (EDAC)
        iv. Hardened memory cells
3. And Recovery Techniques (applied to programmable logic only), such as:
    a) Reconfiguration
    b) Partial configuration
    c) Reallocating design

The first of these groups, the fault prevention techniques, is outside the scope of this thesis as it implies either fabrication process-based techniques or shielding. We will focus on COTS FPGAs, as mentioned earlier, so any technology that implies fabrication will not be discussed. Shielding is an external measure and as such will not be discussed either. The second category of fault tolerant techniques, specifically mitigation, will be the main focus of the thesis. Lastly, the fault removal techniques are also of great interest to us. This kind of methodology is only available to FPGA like devices and plays a great role in the long term robustness of any programmable logic based design.

Fault tolerant techniques are broadly used for ASIC design due to how they can target design architecture without requiring special fabrication methods. As such, many of the fault tolerant techniques are made for non re-programmable devices, like ASICs and do not take into account the special effects of radiation in FPGAs. Fault tolerance techniques can be employed for fault detection (detection techniques) but normally they are applied to both detect and correct system errors in the presence of SEUs or SETs (mitigation techniques). The main principle behind these techniques is that of redundancy, in the form of hardware redundancy, meaning extra

components or in the form of time redundancy, which implies extra execution time or different sampling rates for data. Most frequently a combination of both these types of redundancies is applied for optimal results, yet we will focus solely in hardware redundancy.

3.1.1 Full Hardware Redundancy

Mitigation techniques are, as their name implies, techniques that take effect once the SEU has already happen, preventing it from disturbing the normal functioning of the circuit. As such they tend to be applied with a focus on efficiency, by protecting only critical elements and saving resources, full hardware redundancy is no exception. This type of redundancy can be summarised as logic redundancy, meaning extra logic components or extra paths that allow a design to continue operation even when some of the logic fails. In order to protect a design from the effects of both SEUs and SETs one of the techniques applied is that of TMR. In TMR the logic is triplicated and voters are placed in the output to identify and select the correct value. The most basic form of TMR is that of device TMR, as seen in figure 3.1. This kind of implementation is not resource optimised and allows only for mitigation of any number of faults in a single replica.

In figure 3.2 a better implementation of TMR is shown. Here the solution is applied more efficiently by focusing on a sensitive element of the design, in this case a memory cell, a FF. This type of implementation protects the design, specifically the sequential logic, from the accumulation of SEUs. The way this is intended to work is for the majority voters (MAJ) to detect any errors happening in the FFs between clock cycles, correct them and send them back to the FFs for them to be latched. This method only allows for one upset per FF at a time.

In [10] a TMR design is proposed to protect both combinational and sequential logic while still preventing the accumulation of upsets, figure 3.3. The combinational

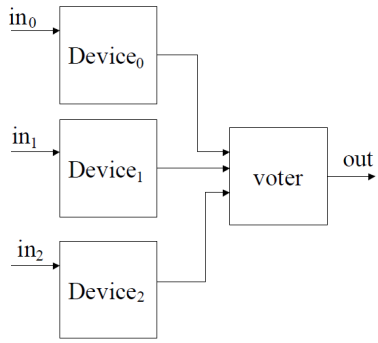

Figure 3.1: Device Protected by TMR [10].

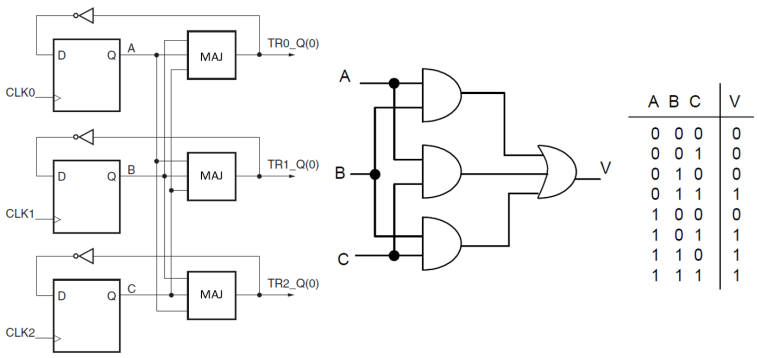

Figure 3.2: Sequential Counter Protected by TMR [10].

logic is triplicated and fed to Multiplexers (MUX) that will ensure the FFs latch the correct value. The MUXs select between the output of the voters or the combinational logic by means of a select input line controlled by a clock frequency established by the designer. This prevents errors that happen between clock cycles from accumulating, since the FFs will have either a new value from the combinational logic or the previous correct value from the voter. Yet again this type of TMR has some limitations and will only prevent failure as long as only one of the branches presents errors at the same time.

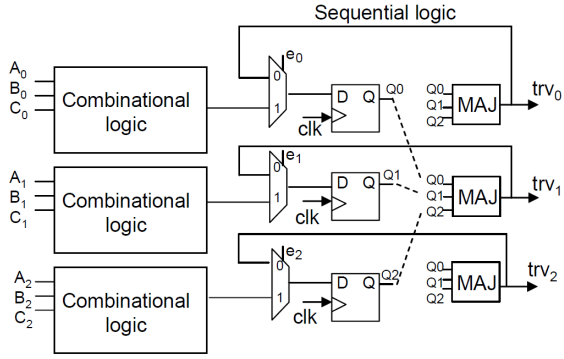

Figure 3.3: TMR for Combinational and Sequential Logic [10].

3.1.2 EDAC

EDACs are another form of mitigation techniques targeted at sequential memory elements. Their purpose is to encode the input of the memory element generating parity bits. These parity bits are then compared to a second group of parity bits generated at the output of the memory element. By doing this several check bits are produced that allow the EDAC to detect if there was an error in the data stored at the memory element. Based on these check bits the EDAC can decide which data bit needs correction, changing its value to the originally intended. A simple EDAC can detect and correct one error at a time within the memory element.

The theory behind the EDAC described is based on the idea of Hamming codes [13]. These codes use the aforementioned parity bits to represent the data bits contained by the memory element. The amount of parity bits required depends on the number of data bits to protect, with a relation shown in the following equation:

$$2_n^P \geq D_n + P_n + 1 \tag{3.1}$$

Where (D_n) represents the number of data bits and (P_n) the number of parity bits. The combination of parity and data bits forms what is referred to as the code word. This code word follows a certain structure, with parity bits placed in location numbers based on the power of 2 (2^0,2^1,...,2^{P_n}), with the least significant parity bit, (P_0) being placed at 2^0 and the most significant, (P_n), at 2^{P_n}. Thus the structure of a 12 bit code word protecting an 8 bit register, which needs 4 parity bits ($2^4 \geq 8+4+1 = 16 \geq 13$), will be like the one shown in figure 3.4.

Each of the parity bits represents a certain number of data bits based on its position. For instance parity bit (P_0) will represent data bits (D_0, D_1, D_3, D_4, D_6...). The way of looking at this logically is by starting at the position of bit (P_0) skipping a number of bits equal to its position and then taking into account a number of bits also equal to its position. So in the case of bit (P_0) being in position 1 we would skip 1 bit, then take 1 bit, then skip 1 bit,...etc, until the end of the code word is reached. Once again using the 12 bit code word in figure 3.4, the bits represented by parity bits (P_{0-3}) will be as shown in figure 3.5.

This relation between parity bits and data bits is characterised as Exclusive OR (XOR) behaviour. So the equations for each of the parity bits in the 12 bit code word

Position	12	11	10	9	8	7	6	5	4	3	2	1
Code Word	D_7	D_6	D_5	D_4	P_3	D_3	D_2	D_1	P_2	D_0	P_1	P_0

Figure 3.4: Code Word for an EDAC(12,8).

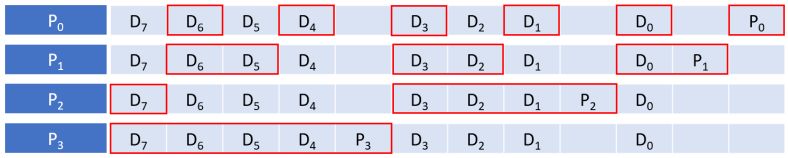

Figure 3.5: Data Bits represented by Parity Bits in an EDAC(12,8).

are as follows:

$$P_0 = D_0 \oplus D_1 \oplus D_3 \oplus D_4 \oplus D_6$$
$$P_1 = D_0 \oplus D_2 \oplus D_3 \oplus D_5 \oplus D_6$$
$$P_2 = D_1 \oplus D_2 \oplus D_3 \oplus D_7$$
$$P_3 = D_4 \oplus D_5 \oplus D_6 \oplus D_7$$

Once the parity bits are generated at both the input and the output of the memory element, they are compared to generate the check bits. This comparison also follows XOR behaviour.

$$C_n = P_n(in) \oplus P_n(out) \tag{3.2}$$

Using the check bits, the EDAC can detect errors in the data bits following the truth table shown in figure 3.6. This is then used to decide which data bits need correction. A full diagram representing the entire system can be seen in figure 3.7.

C_3	C_2	C_1	C_0	Error in Data Bit
0	0	0	0	None
0	0	0	1	Parity bit
0	0	1	0	Parity bit
0	0	1	1	D_0
0	1	0	0	Parity bit
0	1	0	1	D_1
0	1	1	0	D_2
0	1	1	1	D_3
1	0	0	0	Parity bit
1	0	0	1	D_4
1	0	1	0	D_5
1	0	1	1	D_6
1	1	0	0	D_7

Figure 3.6: Errors in Data Bits Detected by Check Bits in an EDAC(12,8).

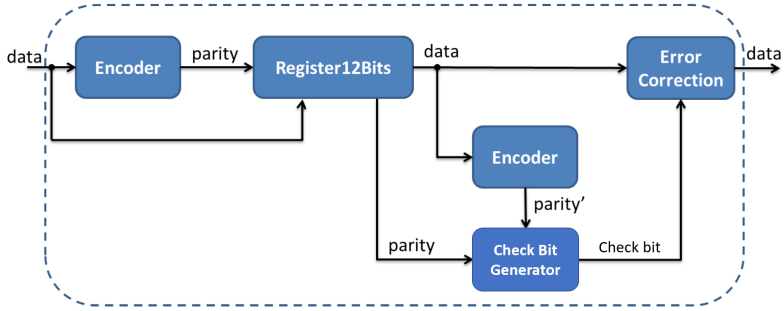

Figure 3.7: Flow diagram for an EDAC(12,8).

3.1.3 Reconfiguration and Reallocation

Recovery techniques follow a completely different approach to increasing robustness when compared to mitigation techniques. Recovery, as the name implies, describes any technique that takes place once the error has occurred and subsequently allows the system to recover from it. In the case of none field-programmable devices recovery would imply replacing damaged parts or components for new ones, but in the case of devices such as FPGAs , techniques like reallocation can be deployed. Reallocation is a technique that targets the design implemented in any FPGA like device, with intentions of re-implementing it on another part of the device that has not been affected by permanent faults. Another similar technique that can be applied for system recovery in FPGA like devices, is that of reconfiguration. Here the device would be reprogrammed in order to fix errors that can't be mitigated and are not permanent. The details on recovery techniques will be described in section 3.4.3, after a more in-depth background on FPGAs has been provided for proper understanding of such techniques.

3.2 The Peculiar Effect of Radiation on SRAM-based FPGAs

Unlike in ASICs, SEUs have a peculiar effect in FPGAs. When a particle hits the combinational logic in an ASIC, it is interpreted as a transient pulse and depending on its duration it might or might not be latched by a storage cell. When the fault occurs in the sequential logic, it is referred to as a bit-flip and will remain in the storage cells affected until their next load [10]. The difference within FPGAs lies in all the sequential and combinational logic being implemented in Configurable Logic Blocks (CLB) by customizable logic memory cells, SRAM cells. This is known as the configuration memory. A way of looking at this is by segmenting the FPGA in three

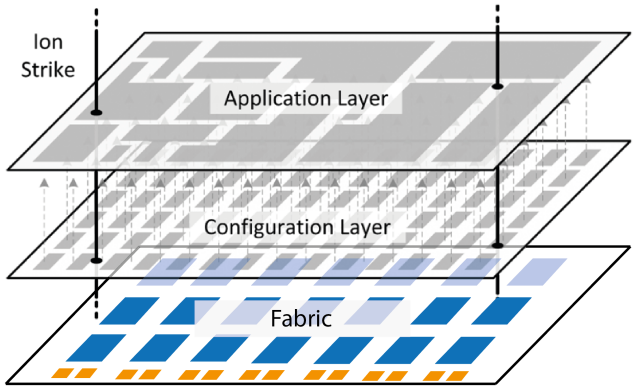

Figure 3.8: Conceptual Layers of an FPGA [8].

layers, the application layer, the configuration layer and the fabric, as seen in figure 3.8. Both the application layer and the configuration layer are abstractions of the fabric, the first being the design described in Hardware Description Language (HDL) and the second the configuration establishing this design in the fabric. The fabric is the only real physical layer, containing the CLBs plus all the other elements characteristic of an FPGA architecture. When an upset occurs in the combinational or sequential logic designed in the application layer, it will in reality affect the fabric. Since this physical layer is controlled by the configuration memory, the vast majority of the errors that happen will permanently reconfigure the design until the bitstream is updated and the values of the configuration memory are reestablished to the intended ones.

In figure 3.9 we can see the configuration layer. Here an arrangement of logic bits

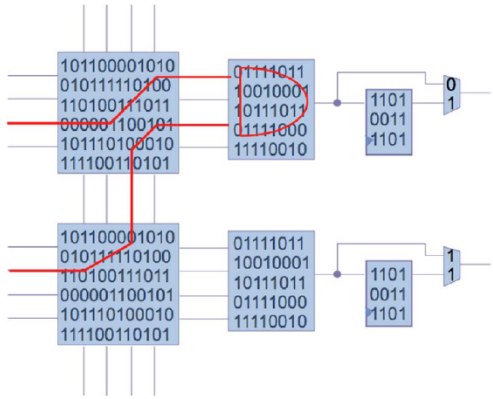

Figure 3.9: Routing and Logic Before Upset [4].

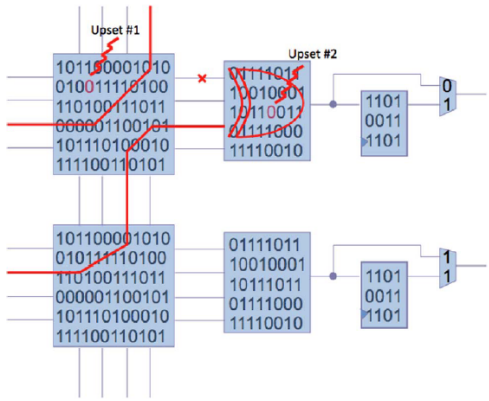

Figure 3.10: Routing and Logic After Upset[4].

in the SRAM cells are configuring the routing and functioning of the fabric, in order to achieve the design specified in the application layer. The design shown here is that of a simple AND gate. If an upset was to occur in the routing logic within the fabric, it might affect the configuration bits establishing the connections. This in turn could result in one of the inputs of the AND gate getting disconnected. Furthermore, if an upset happened within the Look Up Table (LUT) responsible for the AND gate behaviour, its configuration bits could change and the physical LUT in the fabric could start behaving as a XOR, or any other gate. In this case this would happen by permanently updating the results for the input combinations of the LUT. Both this events can drastically change the functioning of the digital circuit, as can be seen in figure 3.10.

Not all impacts from radiation will actually affect the design. If a SEU takes place in the part of the fabric where none of the elements are being used, the effects of the SEU will still be felt by the FPGA but the design wont be affected and will continue

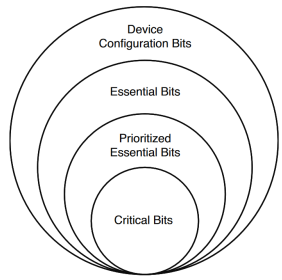

Figure 3.11: Classification of Configuration Bits by Criticality.

functioning as normal. Furthermore, even in the case that the part of the fabric affected is being used by the design, the configuration bit affected by the SEU, in this case known as Essential Bits (EB), could still not disturb the normal operation of the design. Configuration bits that disturb the functioning of the design when affected by SEUs are called Critical Bits (CB). A relationship that explains the concept of bit importance can be seen in figure 3.11.

3.3 Architectural overview of 7-series Xilinx FPGAs

Given the insight into the effects of SEUs in FPGAs presented in the last section, some clarifications as to what the fabric of the FPGA is actually composed of are required. The architecture of the 7-series FPGAs can be described as a column based Advanced Silicon Modular Block (ASMBL) architecture [20]. This architecture changes from family to family, yet the differences are minor and reserved to only some changes in the column fabric related to the number and type of columns available. For simplicity purposes we will use a simplified model of the Xilinx FPGA architecture in order to describe its main components, figure 3.12.

In the figure we can observe the center of the chip being surrounded by Input and Output Blocks (IOB), this elements of the fabric are what allow the chip to communicate with the external world. At the center of the chip the CLBs can be found. These blocks are the main logic resource for implementation of sequential as well as combinational circuits and thus essentially contain the designed digital circuit. Both the IOBs and CLBs are connected by customizable routing resources, commonly referred to as the Interconnect Resources (IR), generally consisting of wire segments and Switch Matrices (SM) made up by Programmable Interconnect Points (PIP) [12]. The fabric will also have other silicone based elements such as Block RAMs (BRAM) or Clock Management Tiles (CMT) that will still be managed by the configuration memory, just like the IRs, the IOBs and CLBs. A more detailed view of the complex 7- series architecture can be observed in figure 3.13.

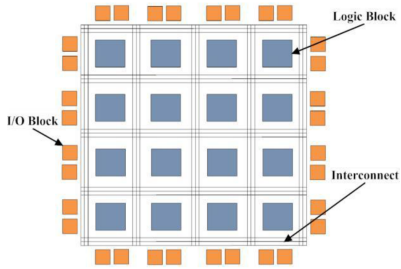

Figure 3.12: Simplified FPGA Fabric[1].

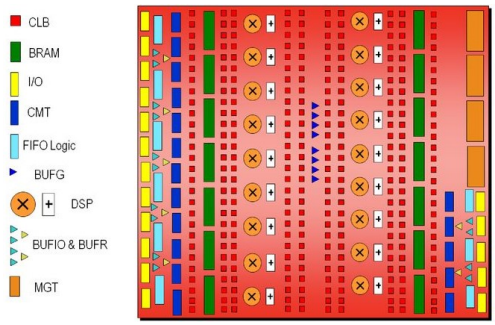

Figure 3.13: 7 series FPGA Fabric.

IR in Xilinx 7-series FPGAs As mentioned before the IR structure is mostly composed of wire segments and SMs. The SMs are considered a key routing structure between the CLBs [12]. It is through these SMs that the FPGA can achieve flexibility in its implementation of the design. The SMs are composed of PIPs consisting of one or several transmission gates separately controlled by configuration bits. There are three types of PIPs present in FPGAs, seen in figure 3.14, the Break-point PIP, the Cross-point PIP and the multiplexers (MUXs) [16]. Another element present in the IR is the buffer. Buffers are similar to PIPs with the exception that they only connect elements in a specific direction. The totality of the IRs can take up to 80% of the available space depending on the FPGA and is referred to as the global IRs. The individual SM connecting to each CLB, as seen in figure 3.15, are considered local IRs.

CLB in Xilinx 7-series FPGAs Each of the Xilinx 7-series CLBs is composed of two slices and connected to a SM for access to the global IRs, figure 3.15. Both slides individually consist of four 6-input LUTs plus their eight FFs, multiplexers and arithmetic carry logic. The LUTs as well as the FFs can carry out different functions depending on their configuration bits and the arrangement of all the elements within the CLB depends on the configuration memory. The only exception would be that

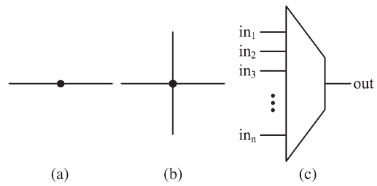

Figure 3.14: Types of PIPs. (a)Break-point PIP.(b)Cross-point PIP.(c)MUX..

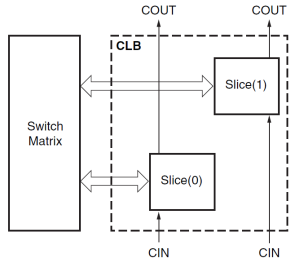

Figure 3.15: Slices within CLB.

of the internal state of the FFs, which are independent, with their values changing based on the design and its clock cycles.

Xilinx's System on a Chip (SOC) devices Given our board of choice, the zybo-z7020 with a ZYNQ SoC FPGA, a brief explanation of the particular architecture of SOC FPGA devices has to be given. These FPGA chips are composed of two different parts, the Programmable System (PS) and the Programmable Logic (PL). The PS is a silicon implemented microcontroller, in our case an ARM. The PL is the part of the chip characteristic of any FPGA and in our case it is equivalent to the Artix7 family of FPGAs. The combination of this to parts is what constitutes a SOC, as can be seen in figure 3.16. The PS in the zybo-z7020 is responsible for UART communication with the computer, meaning that all designs implemented in the zybo that use UART will have to be connected to the ARM by means of a block design. With the ARM itself being programmed in C through Xilinx's SDK platform

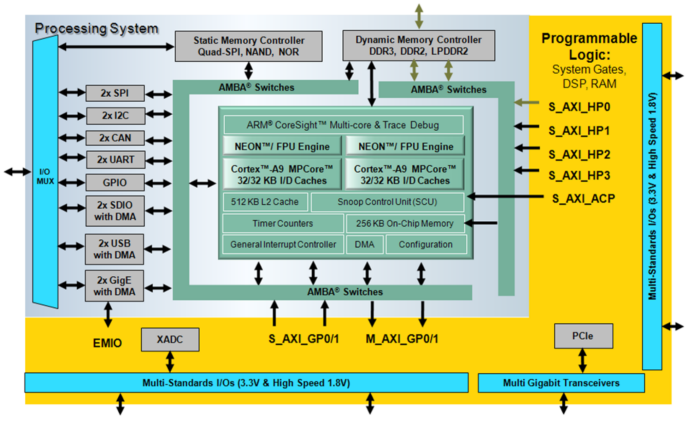

Figure 3.16: Xilinx SoC Architecture.

[14].

Based on this general description of the Xilinx 7-series architecture, the contents of the configuration data for an SRAM FPGA define the values held in LUTs and RAMs, the interconnection between resources, and the modes of the resources [7]. With the exception of the internal state of FFs and the PS

3.4 SEU Mitigation and Recovery for FPGAs

When design based mitigation techniques are applied to digital circuits inside FPGAs, we say the mitigation is being implemented at the High-Level description. Referring back to figure 3.8 this high level description is the application layer, so any mitigation technique applied here will inevitably be implemented in the fabric by the configuration memory, just as usual. This poses a great problem for FPGA designs with SEU mitigation, due to mitigation techniques only preventing errors as long as their configuration bits are not affected by ionizing radiation, something that is bound to happen in aggressive environments such as space. In order to supplement mitigation techniques applied in FPGA designs, recovery techniques have to be deployed alongside them. Recovery techniques make sure that the configuration bitstream remains the same as the originally intended, fixing any errors in the configuration bits. A flowchart analysis of the different mitigation and recovery techniques can be seen in figure 3.17.

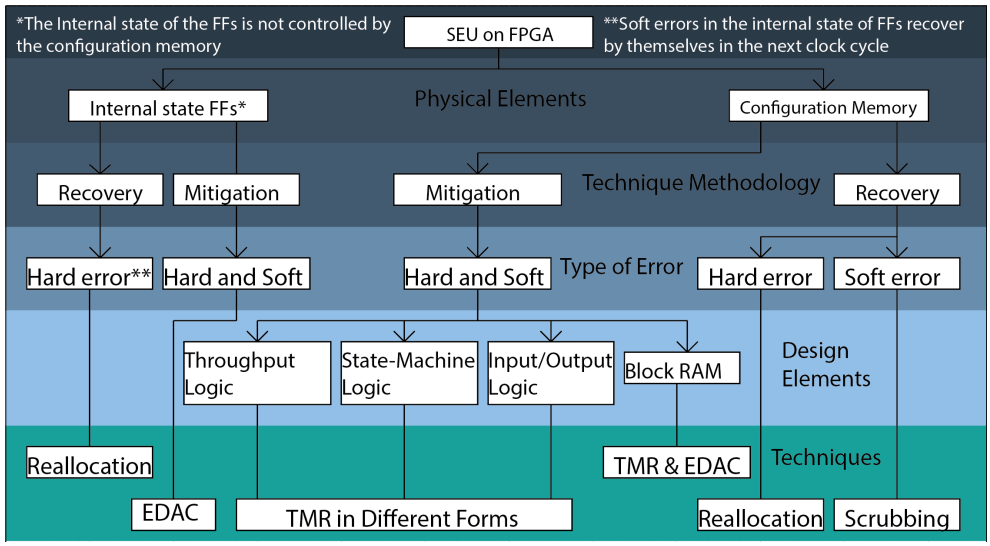

Figure 3.17: Top Down Flowchart for Fault Tolerant Techniques.

The flowchart shows a top-down classification of the different fault tolerant techniques that can be used. Given the nature of FPGAs and their configuration memories, the first important distinction is that of the elements from the fabric that don't depend on the configuration layer. In the case of Xilinx 7 series FPGAs the only elements not dependent on the configuration memory are the Flip Flops (FF), particularly their internal state. Once the physical element distinction has been made the type of fault tolerance methodology to be followed has to be selected. When following the recovery methodology it is critical to consider the type of error, as they will require different treatments. When following the mitigation methodology the most important distinction lays in the design elements, as the way to mitigate errors in them will differ. In order to know which techniques to apply depending on the design its important to know the physical elements used by the FPGA when implementing the design, as mentioned in section 3.3.

3.4.1 TMR in FPGAs

In the case of following a mitigation methodology for FPGA designs implemented by the configuration memory, TMR is the most common technique to be applied. The ways TMR is applied vary in each design, but a general classification of the design elements where it will be deployed at a small scale can still be made and extrapolated to general designs. This classification encompasses three basic types of design elements that are present in most, somewhat complex, FPGA designs. These design elements are Throughput Logic, State-Machine Logic and Input-Output Logic. The first describes any circuit where the output only depends on the inputs and has no feedback loop. The second describes circuits that have loops and depend on different stages of a pipeline. The third describes the connections to the physical inputs and outputs of the FPGA.

In order to demonstrate the differences in TMR, a simple design was carried out to encapsulate the mitigation of SEUs in Throughput Logic and State-Machine Logic, figure 3.18. The design here would be a real application of the theoretical concept presented in section 3.1.1 figure 3.3. With the addition of a TMR clock divider, figure 3.19, following the theory represented in figure 3.2.

The functioning of this TMR design can be observed in the simulation presented in figure 3.20. Here the waveform shows the behaviour of the different signals. The first signal of interest is that of the DATA signal, this signal is fed directly to the combinational logic blocks which alter it based on typical AND behaviour. The second signal of interest is the output of the FFs which update every clock cycle. During the simulation the impact of a SEU in one of the FFs is replicated by setting the value of its output to '0' when it should be '1'. This kind of SEU induced error is caught by the voters and corrected, as can be seen in the final output, since it shows the correct "111" result. When two of these errors happen at once, in two different FFs, the voter is unable to cope with it and the result signal is driven to zero. A similar behaviour can be seen when simulating a fault in the combinational blocks. When

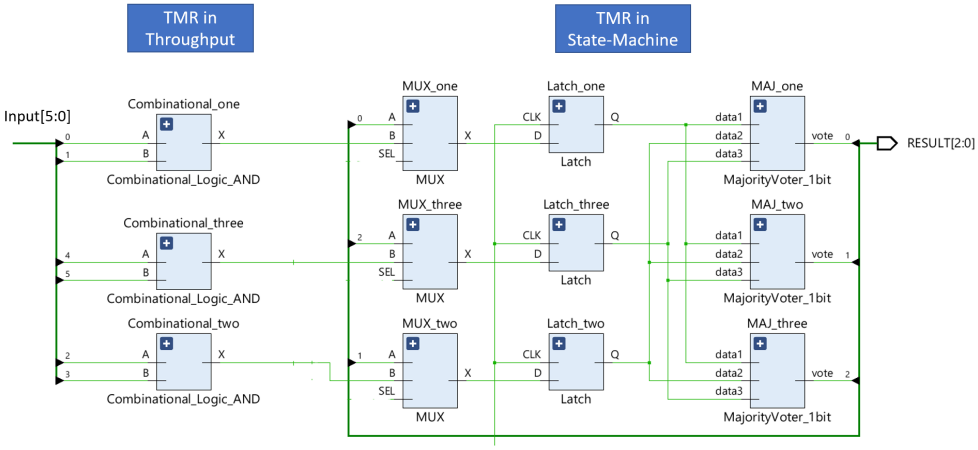

Figure 3.18: TMR Design for a Combinational Block and a FF with Feedback.

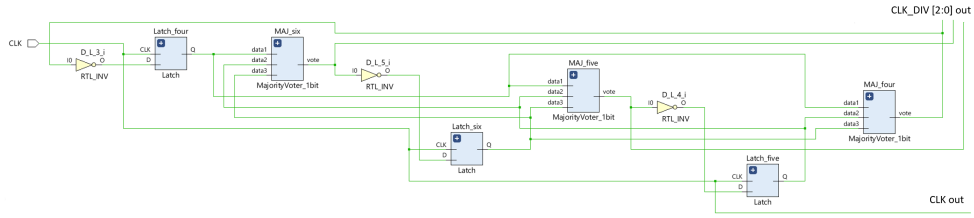

Figure 3.19: TMR Clock Divider.

one of the data lines is forced to a '0', the result signal still maintains its intended "111" value, yet if two data lines experience faults simultaneously the result will be driven to zero as well.

The simulation shows that two errors cannot happen simultaneously in two different lines, no matter if at the combinational or sequential stage, if the result is to be corrected by the voters. Yet in reality this system is capable of dealing with multiple errors due to the MUX select lines feeding the correct value from the voter at a specific frequency. If for instance 2 errors where to occur in the combinational logic while the MUX select line is active, the system will simply just load the previous correct values, hence mitigating the errors. So ultimately the most sensible part of the design are the FFs, since they can only hold one error at a time, independently of the MUX.

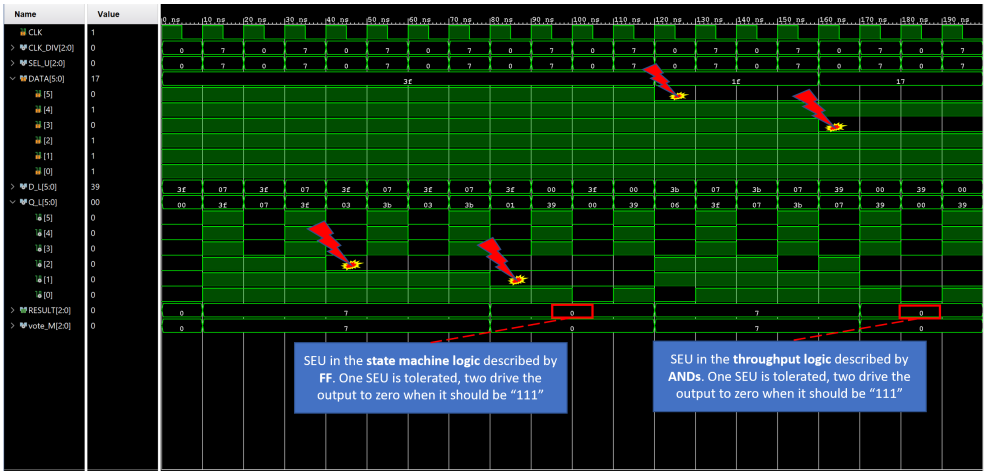

Figure 3.20: Waveform Showing TMR Error Mitigation.

3.4.2 EDAC in FPGAs

In the case of the internal state of FFs or block RAMs all together, implementing an EDAC into their design is the best way to ensure some mitigation. Based on the theory presented in section 3.1.2, a simple design for a 16 register is proposed in figure 3.21.

The design is composed of two encoders that generate the parity bits by means of XOR gates, two registers, the main one being the 16 bit register where the data is stored and the second one holding the values of the first parity bits generated and a correction block composed of MUXs and NOT gates. The purpose of the encoders is to generate a pair of 5 parity bits, one at the input of the 16 bit register and one at the output, in order to allow for comparison and error detection in the correction block. The correction block follows a behaviour that can be described by means of a truth table with the check-bits as its inputs. This truth table shows which bit of the code-word has the error based on the values of the check-bits, following an

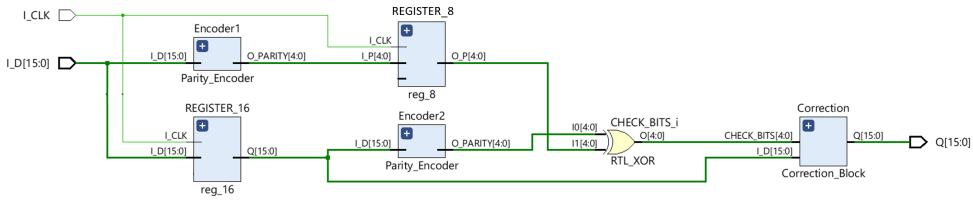

Figure 3.21: EDAC for a 16 Bit Register.

incremental progression where ”00001” represents position ’0’ in the code word and ”10101” represents position ’20’ in the code-word. Based on the specific value of the check-bits the affected bit is restored to its intended value, as can be seen in the waveform shown in figure 3.22.

In the waveform we can observe how an initial SEU impacts the register changing one of its bits from a ’0’ to a ’1’. This is detected by comparing the parity bits thus generating the specific check-bits that in turn are used to correct the output. As can be seen in the waveform during the initial SEU the output of the design remains equal to the input, thus ensuring the correct functioning of the system during the upset. The same cannot be said for the second SEU simulated, since in this case two SEUs are happening at the same time, the EDAC is unable to cope with it and the output changes according to the bits that are affected.

3.4.3 Scrubbing in FPGAs

From the two previous sections it is easy to see that mitigation will only work as long as faults do not happen in multiple lines at the same time, in the case of TMR, or as long as they do not affect several bits of a register or a combinational block, in the case of an EDAC. So in order to remove this faults before more occur and eventually accumulate, techniques related to the recovery methodology such as scrubbing can be applied.

But, before diving into the specifics of scrubbing it is essential to understand how the FPGA is configured each time we upload a design, as the same method will be also used when scrubbing the FPGA. The most critical part of the FPGA configuration

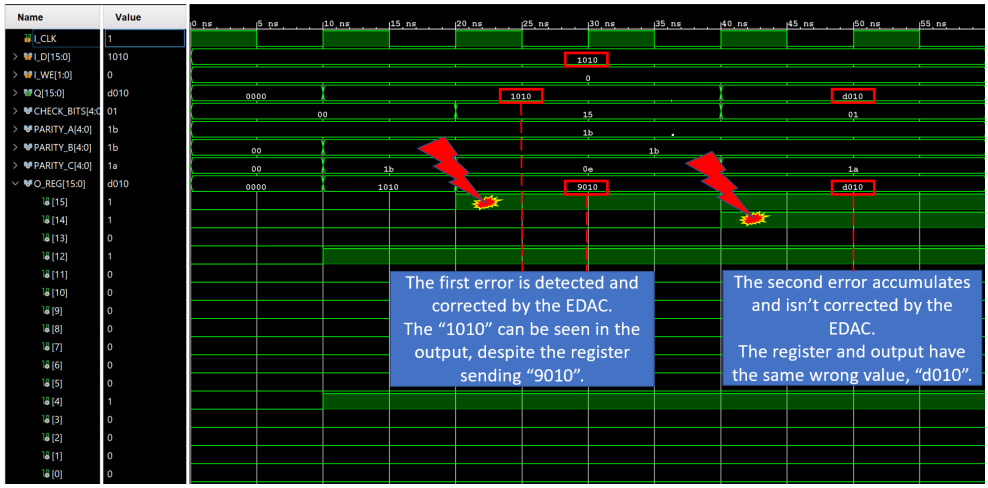

Figure 3.22: Waveform Showing EDAC Error Mitigation.

process is a file generated by the Electronic Design Automation (EDA) tools, known as the bitstream. This file contains all the information necessary to program the specific elements inside the fabric that will be a part of the design described in the application layer. This information encompasses all of the elements described by the end of section 3.3 and represents them by means of one or more bits within the bitstream.

In the case of Xilinx FPGAs the bitstream itself is divided into groups of 32-bit words that are known as frames and are the smallest addressable unit of the configuration memory [22]. These frames have addresses that specify their location within the FPGA that can be broken down into specific segments, figure 3.23. A segment of special importance within the frame address is the block type. It contains the three top bits of the frame address and deals with the different resource components of the FPGA, including the CLBs, CLKs, IOBs and more.

The bitstream itself is organized following a general composition shared by every Xilinx FPGA, with a header, a big chunk of configuration data and a footer. The header is a small section containing commands that prepare the FPGA for receiving the configuration data. The footer comes after the data has been transmitted and sends the FPGA into its startup sequence. The size of the bitstream changes depending on the number of frames it contains, which in turn depends on the number of CLBs in the FPGA.

A specific process has to be followed to load the bitstream into the FPGA. This process is carried out by the configuration module, the sole gateway into the previously described configuration memory layer. In general terms, the configuration module evaluates the bitstream and places its data into the configuration memory. It is through this configuration module that any scrubbing technique will take place.

Scrubbing is normally carried out by specific elements designed for that purpose, either processors or finite state machines implemented in FPGA logic. This elements are referred to as scrubbers and are normally composed of three basic components, figure 3.24: a configuration interface that grants them access to the configuration module, some type of processing logic that is capable of both generating read and write commands as well as interpreting data returned by the configuration module

Address Type	Bit Index	Description
Block Type	[25:23]	Valid block types are CLB, I/O, CLK (000), block RAM content (001), and CFG_CLB (010). A normal bitstream does not include type 011.
Top/Bottom Bit	22	Select between top-half rows (0) and bottom-half rows (1).
Row Address	[21:17]	Selects the current row. The row addresses increment from center to top and then reset and increment from center to bottom.
Column Address	[16:7]	Selects a major column, such as a column of CLBs. Column addresses start at 0 on the left and increase to the right.
Minor Address	[6:0]	Selects a frame within a major column.

Figure 3.23: Frame Address Registers [22].

and a golden memory module containing the correct bitstream [4]. Scrubbers tend to work in a deterministic manner, meaning they will scan the configuration memory frame by frame following a specific order. In the case of a SEU in any element linked to the configuration memory, its correspondent bits will be affected and this will be detected by the scrubber while it is doing its scan. The scrubber will then reload the original bitstream stored in its own memory into the FPGA configuration module that will in turn reprogram the FPGA. This will remove any non permanent faults in the physical elements whose internal values, modes or arrangements are controlled by the configuration memory.

The process of scrubbing is much more complex and detailed that the one described here and as such it escapes the scope of this thesis. This is the reason why no scrubbing techniques will be deployed in the application described in the next chapter. Yet, scrubbing is a crucial aspect of FPGA dependability and had to be discussed nevertheless.

3.4.4 Design Reallocation in FPGAs

Design reallocation is another recovery technique that can be implemented to cope with permanent faults within the FPGA fabric. It follows a similar process to that of scrubbing with the principal difference being the bitstream that is uploaded into the FPGA. In reallocation, the module responsible for detecting a fault in the configuration memory will also have to detect whether this fault is permanent or not. This could be done by seeing what faults persist after an initial scrubbing process. Once a permanent fault is detected, the module would recognise the configuration memory frame affected. Using the block type data contained in the affected frame, the reallocation module will be able to find out which logic element is permanently damaged. Knowing this the module will then proceed to rewrite the bitstream, with the intention of implementing the design in a different section of the fabric, where no

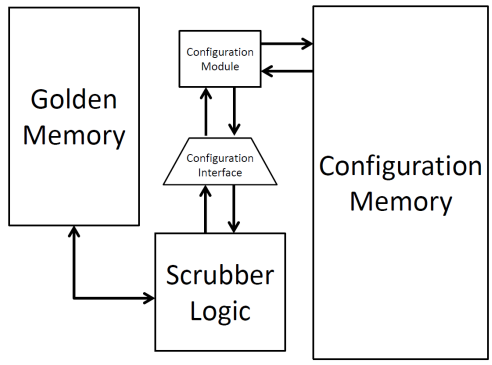

Figure 3.24: Basic Scrubbing Components [4].

damage has been taken.

This is an even more complex process that scrubbing and as such it will not be implemented in the application either. It is also worth commenting that there is a lot of research being done for advanced reallocation techniques that can maximise the efficiency of the process, so the information presented in this section is a vast simplification meant as an introduction to the concept and nothing more.

CHAPTER 4

Making and Testing a Dependable and Secure CPU

When considering the subject of dependability for FPGA platforms the idea of protecting an application or design was the intended final outcome. After all, the purpose of any FPGA deployed in space is that of carrying out a specific arrangement of tasks. Yet, this thesis does not present a particular space application, such as image processing or a communication system, in hopes of simplifying the project's approach while maintaining a realistic outlook. In fact, what is presented in this thesis is a basic micro-controller that works with C applications and could in theory carry out all manner of tasks. Furthermore, micro-controllers implemented in FPGAs are fully customizable bringing more and more advantages to the table in the form of improved performance, flexibility and time to market when considering them as SOC devices [10]. Making the choice of a microcontroller as the target design a sensible one.

4.1 A simple CPU

In order to cut on development time and avoid overstretching the scope of the thesis, it was decided that an already existing micro-controller design will be used, rather than developing one from the grounds up. The design chosen was a basic CPU developed in a lecture by Dr. Juergen Sauermann [14]. The CPU has a similar instruction set to that of the 8-bit CPUs developed by Atmel, specifically an AVR. The instruction set implemented in the CPU is sufficient to run most C programs, meaning that it can execute our custom workloads. The selection process for the CPU was mostly based on the documentation available and the toolchain provided for generating the memory contents of CPU's program memory. On the technical aspect, priority was given to simple CPUs that had a well known instruction set and were at least 8-bit in size.

The chosen CPU consists of the CPU core and a external input/output unit (INO) at the design top level, with the CPU core being the actual AVR. This CPU core

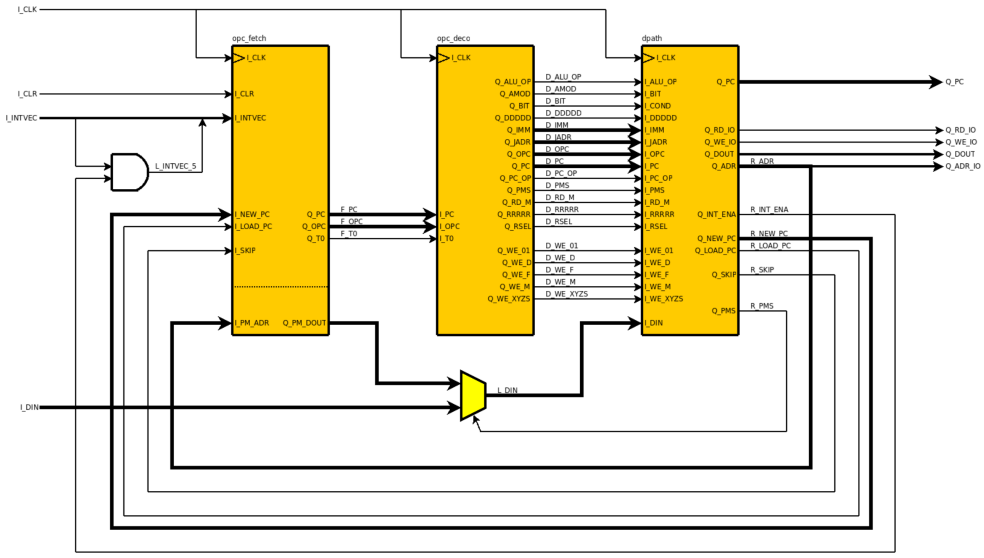

Figure 4.1: Block Diagram of CPU Core [14].

is a three stage pipeline formed by an opcode fetch, an opcode decoding and an execution stage, figure 4.1. The fetch stage contains the program memory and drives the Program Counter (PC) and opcodes alongside other signal to the decoder stage. This second stage decodes the opcodes and generates control signal accordingly for the execution stage.

Some changes had to be made when implementing the CPU in our zybo-z7020 FPGA. For instance, the reset signal for the AVR was also used as the control input for the MUXs, meaning some synchronization errors were intrinsic to the design. In order to avoid these meta-stability and synchronization errors, a register was added at the reset input. Another register was also added at the AVR output in order to hold the data going out. Since the CPU was initially designed for a Spartan 3 FPGA some special considerations had to be made with regards to the ARM CPU in our zybo-z7020 FPGA, since it is through this PS that the UART is driven. The ARM's GPIO was connected to the inputs and outputs of the AVR, with the intention of using this SOC as a control unit. It is through the ARM that the communication with the AVR and the external world is carried out. In figure 4.2 we can see the design including the ARM CPU (Zynq7 PS) and other blocks.

4.1.1 Preparing a Workload

Having a functioning CPU is not of much use if there is no application to run on it, specially if the reason to get it running in the first place is to test its dependability.

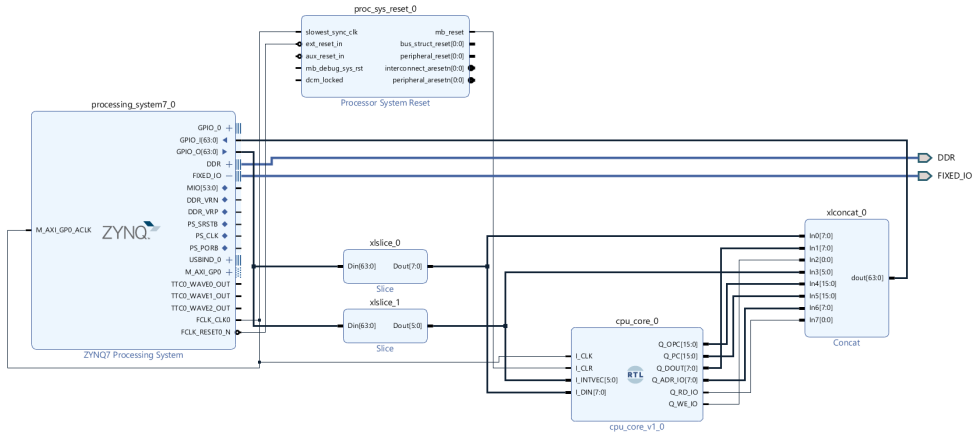

Figure 4.2: Block Diagram of Design Top-Level with ARM CPU.

This is why a workload was designed to represent a realistic use of most of the CPU's resources, with results that can be tested and compared. The workload chosen for the AVR was a basic matrix multiplication, an operation that is representative image processing, a common application in space systems. The workload sends the result from the matrix multiplication to the arm through the outputs of the AVR. The result should always be the same since it was pre-calculated and the matrices never change. The workload was programmed in C and was loaded into the program memory of the AVR by means of a tool-chain provided in [14].

With the workload configured into the AVR's program memory the functioning of the AVR could be tested by running a behavioral simulation in the Vivado suit [14]. Through this simulation it was possible to find out the number of clock cycles that the workload required for completion as well as if the calculations where done correctly by checking the results within the AVR's data memory. Once the functioning of the AVR was demonstrated through simulation it was implemented and simulated post-implementation to ensure it will run perfectly on the zybo-z7020. It was through the completion of this tests that the basic AVR design, before any fault tolerance, was fully developed.

4.2 Fault Injection in the CPU

With the basic AVR design complete with a running workload, it would be possible to test its robustness by means of a method called fault injection. Two different models were used to test the device's robustness, statistical fault injection, for the CLBs and block RAM and simulation based injection, for the internal state of the FFs. These models where both provided by my supervisor Dr. David de Andrés Martínez and

his PhD student Ilya Tuzov.

4.2.1 FPGA-Based Statistical Injection Method

The statistical injection model is based on the process of drawing conclusions for an entire population after conducting a study for a sample taken from that population [17]. In the case of fault injection the conclusions would be the probability of the device failing when injecting faults into its population. The population itself would consist of all the EBs from the configuration memory representing it. Since the two main groups of elements controlled by the configuration memory are the CLBs and BRAMs, these are the two populations that will be tested by the statistical injection method. Since the faults are only injected into the EBs representing the specific population being tested (either CLBs or BRAMs), the probability of failure obtained will represent the percentage of faults within that population that disrupt the functioning of the general design. In our basic AVR design we have 279850 essential bits just for the CLBs alone, so in order to test our entire population we would have to inject faults in each of these bits for each clock cycle, meaning a total of 3917900000 tests to find the exact probability of the design failing according to the CLBs. This would take an unfathomable amount of time and as such the statistical approach is preferred. In order to reduce the amount of tests while maintaining a sensible error margin, the injector is executed with a fixed margin and tests will be carried out until the result is within the pre-established margin. Normally an error margin of ± 0.1 will be set and the injector will run until the probability of the device is within this margin. The statistical injection of faults adds more complexity to a model that in itself is already quite complex. Normal fault injection requires a deep understanding of both FPGA architecture and software programming in order to understand and develop successful models. The model used here was developed

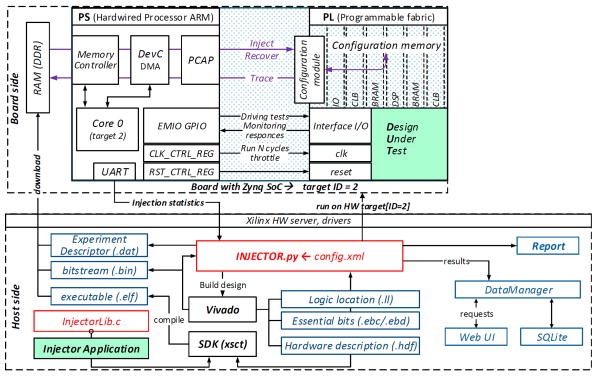

Figure 4.3: Architecture of the fault injector at the software and hardware layers (as provided by the developers).

by Ilya Tuvoz and David de Andrés and consists of a high performance fault injector that targets Xilinx Zynq SoC FPGAs, this being the main reason why the device used in the thesis is a zybo-z7020 FPGA. The fault injector can be fully deployed on-chip and presents many advantages regarding accuracy and minimization of interference with the Design Under Test (DUT).

The fault injector, as explained by the developers, consists of three main components: i) a set of hardware targets (board side) connected to the Xilinx's hardware server (host side), ii) a standalone fault injection application executed on these targets (board side), and iii) an experimentation management application run by the host (host side). The two different sides refer to the platforms needed by the injector, the board being the FPGA where it is implemented and the host being the computer running the scripts and communicating with the FPGA by UART. This complex layout can be seen in figure 4.3.

Board side The board is responsible for running the fault injection, meaning that it is the one executing the workload for the DUT, monitoring and verifying the DUT's responses and toggling essential bits to emulate SEUs. The board also resets the bitstream after each injection, traces de state of the DUT's writable memory and collects the results, reporting the corresponding statistics back to the host. In our case, with the zybo-z7020, all the infrastructure required to execute this tasks is present in the PS.

Host side The host side runs a python script developed by Ilya Tuvoz. This script builds the design in Vivado obtaining the bitstream and corresponding hardware files as well as the essential bits and the logic allocation files. The script also determines the location of the logic cells that should be traced and recovered after injection. These files are used to build an experiment description file that contains all the information required to create a customized template for the DUT. Having the template the script executes SDK and compiles the injector app, which will then be loaded to the device and launched. While the injector runs in the device, the host will start monitoring duties, receiving data and storing it in the database. When the host detects the error margin has been met it will send a command and the injector will stop.

The statistical injector is capable of injecting faults in either the CLBs or the block RAMs, meaning that two tests will have to be carried out in order to obtain accurate results for the DUT's probability of failure. The way to choose where in the configuration memory to inject the faults, as well as other customizable aspects of the injector, is by editing the job description in the python script written by Ilya Tuvoz, shown in Listing 4.1.

```
raw_input("Preconditions fixed, press any key to run the injector >")
        jdesc = JobDescriptor(1)
        jdesc.UpdateBitstream = 1
        jdesc.Blocktype = 1
        jdesc.Essential_bits = 1
```

```
6         jdesc.CheckRecovery = 1
7         jdesc.LogTimeout = 5000
8         jdesc.StartIndex = 0
9         jdesc.Masked = 0
10        jdesc.Failures = 0
11        jdesc.sample_size_goal = 0
12        jdesc.error_margin_goal = float(0.1)
13        jdesc.FaultMultiplicity = 1
14        jdesc.PopulationSize = float(14000)*Injector.
              EssentialBitsPerBlockType[jdesc.Blocktype]
15        jdesc.Mode = 100    #Very custom injection mode (without callbacks)
16        jdesc.SamplingWithoutRepetition = 0
17
18        res = Injector.run(OperatingModes.SampleUntilErrorMargin, jdesc,
              False)
```

Listing 4.1: Job Description Variables in the Python Script.

The Blocktype variable refers to the first address type from the configuration memory frames. By changing this variable we can control which resource to inject faults at. The Esssential_bits variable controls whether the injector will introduce faults into all the bits of the configuration memory or only those used by the design. The error_margin_goal establishes the margin towards which the injector will aim, the lower we set this value the more tests the injector will carry out. Finally, another important variable is the FaultMultiplicity, this variable controls the number of bit-flips that will take place at the same time, meaning the number of SEUs simulated on the same instant in time. By increasing this variable we can simulate harsher radiation environments.

4.2.1.1 The Difficulties of Statistical Fault Injection

The statistical injection model presented above works at the theoretical level, yet when implemented into the FPGA things change. Xilinx does not provide any information regarding the correlation between the bitstream and the elements of the fabric, making it hard to know what is being changed when a bit from the bitstream is flipped. The only exception being the LUTs and BRAMs, as these two elements can be located due to the mask provided in the logic allocation file generated by Vivado. The PIPs on the other hand remain a complete mystery, making changes in the bitstream really hard to trace.

Furthermore, the BRAMS have a silicon-implemented register at the output that resets its value every time the configuration memory is read or written to access the contents of the BRAM. This means that the CPU will read a ”00..00” instead of the actual value intended, in turn causing the probability of failure in the BRAMs to increase drastically, as most faults injected will cause multiple errors that do not correspond to reality. This problem is intensified when triplicating elements containing BRAMs. BRAMs from different instances can often share the same frame in the configuration memory after being allocated into the same column by Vivado’s implementation. When the injector locates a BRAM within a frame and executes a bit-flip,

the effects of this could also be observed in the other BRAMs sharing the same frame. When one of the frames is read or written with hopes of accessing the contents of a specific BRAM, the registers in all of the BRAMs sharing the same frame will be reset, causing the probability of failure to increase far beyond the real measure due to all TMR replicas being read incorrectly by the CPU.

As a result of this problem the BRAM's probability of failure after fault tolerance techniques have been implemented cannot be tested according to the margins established. A simple manual floor-plan can be carried out to ensure that no BRAMs share any frames at all, preventing the number of errors from cascading from one BRAM to another. Yet the problem of the register reset has no apparent solution at the moment of writing this paper, meaning that the results from fault injection will not be accurate for the BRAMs even if they do not share any frames. As an alternative simulation-based fault injection will be used for these elements too.

4.2.2 Post-Implementation Simulation Based Injection

Simulation is a less direct, less intrusive method of finding out the probability of failure of certain elements from the device. By controlling certain signals within the design it is possible to emulate the bit-flips that would occur from SEUs on specific elements. Furthermore, when it comes to finding out the probability of failure for the elements of the fabric that do not depend on the configuration memory, like the internal state of the FFs, simulation is the only option given how the bit-flips in the bitstream from the statistical injection method will not affect the internal state of the FFs. The simulation based injection method provided by David de Andrés and Ilya Tuvoz is basically a collection of custom Python scripts that process incoming XML configuration files and HDL models, to in turn generate the scripts for an off-the-shelf simulator that will: i) run the required fault injection experiments, and ii) generate the required trace files [18]. Given how it takes a considerable amount of computational power to execute the simulations, this process was carried out by the developers (David de Andrés and Ilya Tuvoz) using the university's computer cluster for each of the DUT prepared. This means that the tool did not have to be deployed for the purpose of this thesis, but rather it was used as an external way of testing the designs and as such it will not be described in detail, rather its results will be presented and commented in later sections.

4.3 Assessing the CPU's Dependability

In order to properly asses the CPU's dependability a few things have to be determined, the first being the scenario where the CPU would actually be deployed at. In the same way as matrix multiplication was chosen as a workload representative of what could be a real space application, a scenario will also have to be chosen to represent a realistic near Earth radiation Environment. A good way of doing this will be by

classifying scenarios based on radiation harshness. On general terms two types of scenarios will be discussed, a baseline low radiation scenario where SEUs normally happen in a one at a time basis (simple faults) and a high radiation environment where multiple SEUs can take place in one single instant. The baseline scenario presents faults during the execution of the workload as single events in terms of both space and time, meaning that only one fault will take place during each clock cycle and it will only affect one bit. Whereas the high radiation scenario implies multiple faults occurring in the same clock cycle and affecting several bits. The concept of multiple faults in time was not considered for the purpose of this thesis. Knowing the approximate amount of particles that will impact the DUT allows us to obtain the second element required for assessing the dependability of the CPU, the Probability of Failure (PoF). This measure is obtained from calculations based on fault injection results, and it represents the probability of the DUT failing to operate when faults are injected. In order to obtain a general PoF for the entire DUT the values obtained from the fault injection in each element have to be normalized following equation 4.1.

$$PoF_{Element} = \frac{Fault\ injection\ result\ for\ specific\ element * \mathrm{EB}_{Element}}{\mathrm{EB}_{DUT}} \tag{4.1}$$

The value obtained from equation 4.1 will represent the part of the total PoF that element is responsible for. So in order to obtain the DUT's PoF, the contributions of the CLBs, BRAMs and FFs have to be added.

$$PoF_{DUT} = PoF_{CLB} + PoF_{BRAM} + PoF_{FF} \tag{4.2}$$

The DUT's PoF is used to obtain the Failure in Time (FIT) expressed in FIT units. One FIT equals 1 failure per 1000 million device hours. So for example, 5 failures expected out of 1 million components operating for 1,000 hours have 5 FIT [21]. Normally the FITs are expressed in terms of FIT per megabite, as this would represent the FITs based on the DUT's size. In order to obtain the value for the FITs in units of FIT per Mb, the DUT's PoF has to be multiplied by the Rate of Faults (RoF). The RoF is variable that depends on the target device and the external sources of radiation, specifying the number of faults that will happen for a unit of time (normally hours). The calculation of a DUT's FIT is shown in equation 4.3.

$$FIT_{DUT}(\frac{FIT}{Mb}) = PoF_{DUT}(\frac{\mathrm{EB}_{DUT}}{Mb}) * RoF(h) * (1 * 10^9) \tag{4.3}$$

The FITs in units of $(\frac{FIT}{Mb})$ are normally employed by Xilinx in their dependability assessment documentation, as such all results presented in the following sections will be converted to this units.

If the intended objective is to maximise the designs dependability in any of the scenarios presented, it would be possible to combine several fault tolerant techniques to reduce the FITs to the lowest possible value. Yet, dependability often comes at a cost in FPGA resources and power consumption. So a balance has to be struck between dependability, FPGA resources, power consumption and operating frequency

in order to present the most balanced design for each of the scenarios. In the following sections several techniques will be applied depending on each of the scenarios, always comparing the results with that of the basic AVR design in all of the categories previously mentioned. This will be the methodology followed in this thesis to asses the impact and effectiveness of fault tolerant techniques.

4.3.1 Simple Faults: Low Radiation Environment

The basic AVR has a total of 354038 EB, an on chip power consumption of 1.55 W and a maximum operation frequency of 50 MHz. When injecting simple faults into it a combined PoF of 9.21% ± 0.19 is obtained. The breakdown of this PoF is as follows: i) PoF_CLB = 8.38% ± 0.08, ii) PoF_BRAM = 0.81% ± 0.11, and iii) PoF_FF = 0.018% ± 0.001. The differences in the PoF come from the breakdown of essential bits in the DUT, i) CLBs have 279850 EB, ii) BRAMs have 73728 EB, and iii) FF have 460 EB.

$$PoF_{CLB} = \frac{10.6 * 279850}{354038} \pm \frac{0.1 * 279850}{354038} \tag{4.4}$$

$$PoF_{BRAM} = \frac{3.88 * 73728}{354038} \pm \frac{0.54 * 73728}{354038} \tag{4.5}$$

$$PoF_{FF} = \frac{14.08 * 460}{354038} \pm \frac{0.98 * 460}{354038} \tag{4.6}$$

In units of FIT per Mb the DUT's FITs equate $(3.259 \pm 0.068) * RoF \ \frac{\text{FIT}}{Mb}$.

$$FIT_{DUT}(\frac{\text{FIT}}{Mb}) = (9.21 * \frac{354038}{1000000} \pm 0.19 * \frac{354038}{1000000}) * RoF \tag{4.7}$$

Where RoF remains an unknown that would have to be obtained based on the environment where the design would carry the workload, as well as the target device where the design would be implemented.

Device TMR The FITs for the basic AVR obtained from simple faults can be easily improved by means of mitigation techniques. The first technique that can be used to increase dependability is the simple device TMR. This kind of technique does not require a prior study of the most radiation sensible internal elements of the DUT, since it just triplicates the entire design. The design encompassing this technique has 1017034 EB, an on chip power consumption of 1.57 W and a maximum operation frequency of 50 MHz. This technique aims to reduce the FITs of the design to 0, since we are assuming only one fault happening for each clock cycle (simple fault) and the entire device is triplicated. TMR is applied to the CPU core and three MAJs are used to vote on the 8-bit data out signals to avoid any single point of failure. These three signal are then fed to the ARM, responsible for reading and choosing the correct signal as well as executing the injector app. The entire design can be seen in figure 4.4.

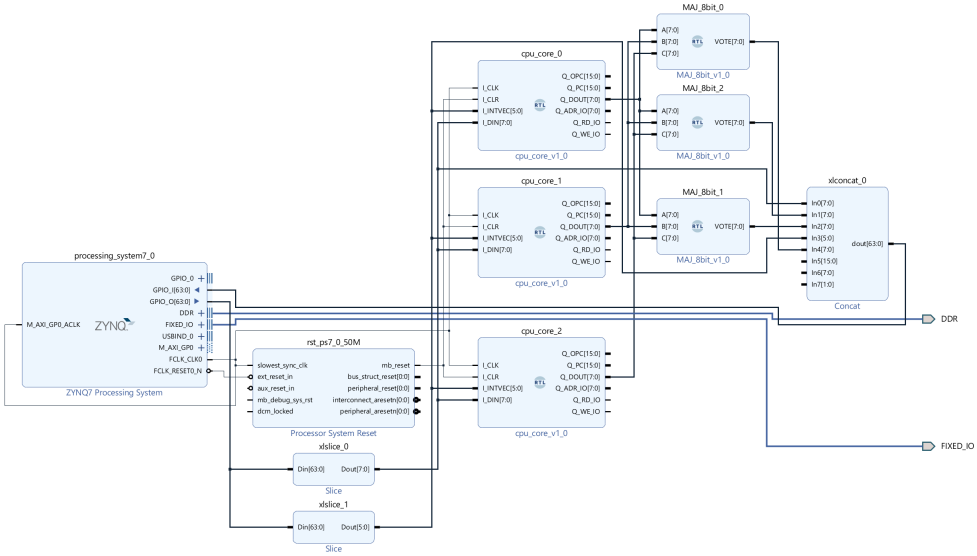

Figure 4.4: CPU Design with Device TMR and Three Voters.

The expected results are met, with 0 $\pm 0\frac{\text{FIT}}{Mb}$ FITs. Yet, this technique implies an increase in area overhead that most systems might not be able to afford (2.87 times the essential bits of the original design), since most professional designs use much more resources than the simple CPU presented here and might run into size restrictions with their devices. This means that triplicating the entire design might not be always an option, or even more importantly it might not always be the most efficient option. The Device TMR will most assuredly also have an effect in the maximum clock frequency of the AVR. As such a few specific instances of TMR can be applied at critical parts of the design, such as the Arithmetic Logic Unit (ALU) or the control unit (OPC_DECO), to in theory achieve similar FITs with much less area overhead, thus increasing the efficiency of the solution.

Simulation based injection is used to asses the radiation sensitivity of different parts of the design for targeted mitigation techniques. Due to the problems with the mapping of the elements of the bitstream, statistical fault injection does not give us any information on the most vulnerable parts of the DUT relating to CLBs. This leaves us with speculation based on the AVR's architecture as the only tool to asses the radiation sensitivity of design elements. Nevertheless, with simulation it is possible to break down the PoFs of the BRAMS and FFs of each DUT to the signals representing them. This in turn allows us to trace our design and see which parts contribute most to the FITs. A breakdown of the FF specific PoF in the basic AVR can be seen in figure 4.5. The percentages shown in the figure represent the number of times the design failed when injecting faults in the FFs alone. Using the

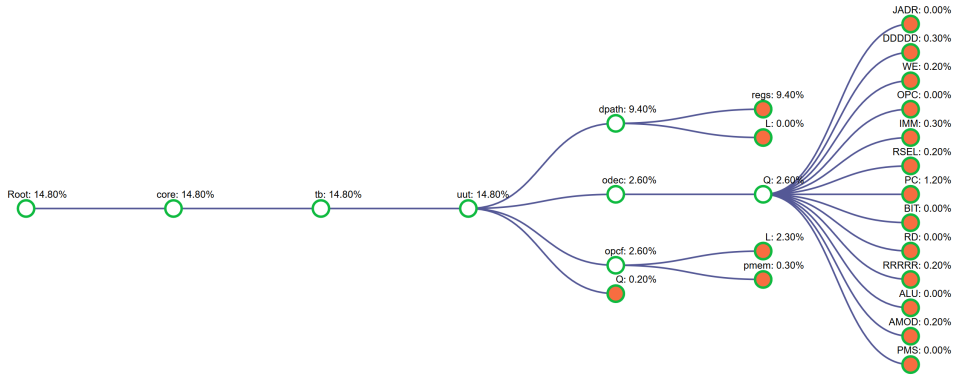

Figure 4.5: FF specific PoF Distribution in Basic AVR.

information presented by the figure we can identify a few critical elements within the design: i) the register file inside the DATAPATH, ii) the registers inside OPC-FETCH responsible for updating the PC, and iii) the OPC-DECO which can be triplicated for maximum results. Yet, when taking the EBs into consideration it is easy to see that any mitigation applied for the FFs will have minimal impact on the PoF of the DUT due to FFs having a basic PoF of 0.018% ± 0.001. Nevertheless, it will still be interesting to see the resource consumption from applying Hamming-based EDAC in this locations. When it comes to the BRAMs the assessment prior to the implementation of fault tolerance is quite straightforward. Since the BRAMS are pre-packaged IP-cores there is no need to speculate or study where mitigation will be more effective, EDACs can just be directly applied to the generated BRAMs.

In order to properly asses the impact each of the mitigation techniques designed can have on the FITs of the three basic elements, the techniques are segmented by their mitigation methodology. Furthermore techniques that target CLBs will be isolated in hopes of tracing the effect they have on the DUT's PoF more clearly.

EDAC in Registers and BRAMs i) Starting with the register file; this element resides inside the DATAPATH and features 16 16-Bit registers as well as a status register. The 16-Bit registers use an enable signal to choose between updating the upper or lower part of the signal, so they can be split into separate 8-Bit registers, each with its own EDAC. This way the system will be able to tolerate 2 faults per 16-Bit register, one in each 8-Bit EDAC register. The 8-Bit EDACs follow the same structure as presented in figure 3.21, with an 8-Bit register for the data bits and a 4-Bit register for parity bits. The inputs and outputs of the EDAC registers are paired to form the final 16-Bit register. The status register is left unprotected for the moment, since its impact on the DUT's PoF was 0.1%. The DATAPATH also includes the "data_mem" BRAM, this memory element is responsible for storing the results of the workload among other data. Since it is a pre-generated IP-core, Xilinx

provides error correction code as an option when customizing it. In the case of the "data_mem" BRAM built in error correction code was used.

ii) The OPC-FETCH component has a register "LPC" responsible for updating the PC "L_PC" and controlling the wait instruction "L_T0". The same 16 bit EDAC used for the register file can be used here to provide double protection to the "L_PC" signal, while a simple TMR can be used to protect the "L_T0" bit. The two "prog_mem" BRAMs also reside inside the OPC-FETCH, yet unlike the "data_mem", these BRAM IP-cores are generated as dual port 16 bit ROMs and have no possibility for built in error correction. In order to provide the ROMs with protection, custom EDAC has to be designed and implemented. Since ROMs have no data input, the design previously used for EDAC cannot be applied, rather two dual port ROMs with the same addresses as the "prog_mems" and 5 bit data outputs will be generated, storing the parity for each memory location. The final design has 4 ROMs total, 4 parity encoders (1 for each port of the "prog_mems") and 4 correction blocks. The "prog_mem" file where the instances of the BRAMs are created also holds a register that updates the PC within the file, as well as two other signals. The PC signal is protected with EDAC and the other two have TMR applied.

Both i) and ii) present mitigation techniques that target registers and BRAMs, with in theory no positive effect on the CLB's contribution to the DUT's PoF. A design encompassing all of these mitigation techniques has a total of 627405 EBs, with 405615 belonging to the CLBs, 221184 to the BRAMs and 606 to the FFs. There is a noticeable increase in the EBs of the BRAMs, reaching the same value as that of the Device TMR. This is due to the addition of the 2 ROMs for parity bits and the error correction in the "data_mem" BRAM. The FFs also see an increase due to the amount of registers generated during synthesis to implement the EDAC designs. The combined PoF obtained from fault injection is $5.47\% \pm 0.06$, with CLBs contributing $5.47\% \pm 0.06$, BRAMs contributing $0\% \pm 0$ and FFs contributing $0.0022\% \pm 0.0009$. The decrease in the CLB's contribution to the DUT's PoF was unexpected, since in theory the addition of logic to mitigate faults in the registers and BRAMs should have increased it. Unfortunately the specifics as to why this could happen are untraceable due to the bitstream mapping. As for the BRAMs the result was as expected, since they where all protected simple faults should be totally mitigated. A few failures remain from the FFs, this is due to some of the registers left unprotected and the OPC_DECO, which will be triplicated in another design. In units of FIT per Mb the combined FITs are $3.435 \pm 0.041 \frac{\text{FIT}}{Mb}$, with the CLB contributing the most. When comparing the FITs with those of the basic AVR, we can see that error mitigation in the FF actually makes the design less robust. Since their original contribution to DUT's PoF was so low to start with, the addition of extra resources (.ergo EBs) just increases the possibility of particle impact without much extra robustness being added, as can be seen in CLB's contribution to the FITs.

TMR in the OPC-DECO iii) The OPC-DECO has both combinational logic and registers, so mitigation techniques targeting it will affect both the contributions

of the CLBs and the FFs to the DUT's PoF. Since TMR has to be applied in the entire component in order to mitigate faults in the combinational logic, there is really no point in designing EDAC for the individual registers as these will be protected against simple faults by the TMR too. The TMR for the OPC-DECO follows a simple structure were the component itself is triplicated and each output is voted on by a single majority voter. In the theory discussed for the TMR as well as in the device TMR designed, the voters where triplicated to avoid single points of failure. Yet based on the results obtained from the TMR in the ALU, table 4.6, where one voter was compared against 3 voters, it became clear that the 3 voters did not improve the PoF enough to justify the extra resources. As such it was decided that one voter would be enough for each signal in the OPC-DECO too. The resulting design has three instances of the original OPC-DECO plus a total of 16 voters, all inside a TMR wrapper that substitutes the original file to maintain a clear hierarchy. The results for the DUT's PoF and FITs can be seen in table 4.6. Even thought the TMR in the OPC-DECO didn't protect any BRAMs it still has an effect on their contribution to the DUT's PoF, since now their percentage in the DUT's EBs is smaller. The contribution to the DUT's PoF of both the CLBs and FFs is smaller, as was expected. Yet when observing the FITs we can see that the improvement on the original design is not too big.

TMR in the ALU This last mitigation technique focuses only on improving the contribution of CLBs to the DUT's PoF by implementing TMR in the ALU within the DATAPATH. The TMR designed follows the standard methodology also applied in the OPC-DECO. The only difference being that this design was also tested with 3 voters, as mentioned in the last paragraph. The CLB's contribution to the DUT's FITs with one voter was 2.454±0.045 $\frac{\text{FIT}}{Mb}$, whereas the one for 3 voters was 2.467 ±0.045 $\frac{\text{FIT}}{Mb}$. The design with three voters had more EBs for the CLBs and this can be seen in the bigger number of FITs. It is clear from this comparison that the use of 3 voters is counterproductive and as such the design implemented will only have one voter per ALU output signal. The design with one voter has a total of 521690 EBs, with 447502 belonging to the CLBs, 73728 to the BRAMs and 460 to the FFs. Since the TMR didn't target any BRAMs or registers, the EBs related to the BRAMs and FFs remain the same as in the basic AVR. When injecting faults the total PoF will be 5.53% ±0.20, with its value in FIT per Mb being 2.885 ±0.104 $\frac{\text{FIT}}{Mb}$ FITs. Just by protecting the ALU we can already see results close to half of the original FITs, with an increase in resources of 1.45 times the original EBs.

TMR and EDAC None of the individual fault tolerant designs achieve the results of the device TMR. Yet, this was an expected outcome, the real question lays in whether the combination of all these designs in to one single overhaul of mitigation techniques will match the device TMR in FITs while consuming less resources. All of the mitigation techniques previously discussed where self contained designs that maintained the same port structure, so they can easily be merged into one single fault

tolerant AVR design. The results from this design, including the EBs, can be seen in table 4.6. Improvement can be observed in the FITs of the design, yet once again the increase in CLB resources to protect the FFs ends up undermining the efforts to reduce the CLB's contribution to the FITs. In the end a much more efficient implementation of fault tolerance would have targeted the BRAMs and CLBs, since their presence in the AVR's design is much bigger in comparison to that of the FFs.

Comparing the Results for Simple Faults Based on the results shown in table 4.6 the FF's contribution to the FITs in all of the designs is extremely small. Yet, the same cannot be said for the effect of utilizing resources to protect the FFs, as it has tremendous impact on the CLB's contribution to the FITs. In general the designs that for the most part, implement mitigation in the registers, like the TMR DECO and the EDACs, show either very little improvement on the baseline FITs or in the case of the EDACs, reduced design robustness with a higher number of FITs. Looking at the results it is clear that choosing where to apply mitigation based on element-specific PoF, like the one shown in figure 4.5, can be misleading. Rather the number of EBs in each population should be compared as the initial indicator of where to apply mitigation techniques. For example in our AVR the CLBs have the most EBs, so techniques targeting these elements will be more effective in reducing the DUT's FITs, since CLB's are its greatest contributors. Yet, this cannot be the only means of measuring where to apply mitigation. If an element can be protected at very little resource expense, the FIT will improve without consequences. Also in the case of FFs, even if protecting them affects the FITs, they still should be protected as best as possible since scrubbing wont be able to remove their faults periodically.

The maximum operating frequency is the same for all designs because the timing constraints where not pushed in order to achieve the fastest operating frequency possible in each design due to lack of time. As for the on chip power consumption, most designs have the same with only the two bigger designs consuming a bit more due to the increase in resources used.

Design (1 FAULT)	EBs Used				Chip Pow. Cons. (W)	Max Op. Freq. (MHz)	PoF (%)				FIT / RoF (FIT/Mb)			
	CLB	BRAMs	FFs	Total			CLB	BRAMs	FFs	Total	CLB	BRAMs	FFs	Total
Basic AVR	279850	73728	460	354038	1.55	50	8.38 ± 0.08	0.81 ± 0.11	0.018 ± 0.001	9.21 ± 0.19	2.966 ± 0.028	0.286 ± 0.040	0.006	3.259 ± 0.068
Device TMR	794569	221184	1281	1017034	1.57	50	0	0	0	0	0	0	0	0
TMR ALU	447502	73728	460	521690	1.55	50	4.70 ± 0.09	0.55 ± 0.08	0.012 ± 0.001	5.26 ± 0.16	2.454 ± 0.045	0.286 ± 0.040	0.006	2.746 ± 0.085
TMR DECO	392713	73728	659	467100	1.55	50	6.28 ± 0.08	0.61 ± 0.09	0.016 ± 0.001	6.91 ± 0.17	2.932 ± 0.039	0.286 ± 0.040	0.007 ± 0.001	3.226 ± 0.080
EDAC in BRAMs & FFs	405615	221184	606	627405	1.55	50	5.47 ± 0.06	0	0.002 ± 0.001	5.47 ± 0.07	3.434 ± 0.041	0	0.002 ± 0.001	3.435 ± 0.041
TMR & EDAC	667775	221184	812	889771	1.57	50	2.55 ± 0.08	0	0	2.55 ± 0.08	2.270 ± 0.067	0	0	2.271 ± 0.067

Figure 4.6: Comparison of Resources, PoFs and FITs for the Different Designs (Simple Faults).

4.3.2 Multiple Faults: High Radiation Environment

All the results presented previously were obtained from simple fault injection, simulating what would be a low radiation environment. But in order to test the design's performance in what would be a high radiation environment, more faults have to be injected in the same clock cycle. The number of faults to be injected would normally be estimated based on the environment where the device would execute the workload. But since there is no specific data for any environment, 4 faults will be injected per clock cycle to simulate harsh radiation. This of course will increase the FITs of the designs previously presented, obtaining uninteresting findings from most of them. Yet, a few of the designs use techniques that in theory should hold up to more than just simple faults, these are the designs that will be injected with 4 faults.

Device TMR Given the harshness of the scenario the basic AVR will show a catastrophic number of FITs, so rather than using this as the baseline design the device TMR will be used instead, as it was the most robust design when injecting simple faults. The results from injecting 4 faults can be seen in table 4.7, where the increase in FITs for the device TMR is quite harsh. The contribution of the CLBs to the FITs is still the biggest, yet since now the 6 BRAMs are affected by more than simple faults, their contribution is also quite important. The FFs still have a minuscule contribution. If EDACs where to be implemented in the BRAMs, their contribution to the FITs would decrease drastically.

TMR and EDAC In order to test the theory about the implementation of EDAC in the BRAMs being able to decrease their contribution to the FITs, the design encompassing all of the targeted mitigation techniques is tested with 4 faults too. The results can be observed in table 4.7, where our predictions are confirmed. The EDAC for the BRAMs are composed of two individual EDAC registers, as previously explained, and as such they can tolerate 2 faults per BRAM. Since only 4 faults are injected the probability of more than 2 faults affecting the same word in the same BRAM is extremely small, given their number of EBs. Yet, even when the contribution of the BRAMs is zero, this design still employs a considerable number of resources in protecting registers, thus increasing the contribution of the CLBs and in turn yielding a number of FITs worse than that of the device TMR.

Device TMR for a CPU Core with TMR and EDAC With hopes of reverting the negative effect of protecting the registers in the previous design, the entire CPU core encompassing the TMR and EDAC designs was triplicated. This would yield the most resource expensive design, but in theory it should be more robust, since it triplicates elements that were already internally triplicated. Yet, from the results in table 4.7, it can be seen that in fact the opposite is true. Starting with the CLB's contribution to the PoF, we can already see that design does not gain much robustness. When time is brought into the equation, in the form of FITs, the results are catastrophic, with the highest number of FITs of all the designs.

Comparing the Results for 4 Faults From the results obtained it is clear that aimlessly triplicating elements will not necessarily improve robustness, rather it might affect negatively due to the increase in surface area. In the case of the device TMR for the AVR the most effective mitigation would have been adding EDAC to the BRAMs and TMR to the ALU, while leaving the registers unprotected. Of course this is due to the nature of FPGAs with their SRAM-based configuration memory composing the majority of the population, FFs only being a minor concern based on their EBs. Yet as mentioned before, scrubbing wont protect FFs allowing faults to accumulate over time. One could argue that getting their contribution to the PoF as close to 0 as possible is a mayor necessity.

4.4 Conclusion

When looking at all the results the biggest contribution to the FITs are the CLBs, since they are the biggest population in all the designs tested. This leads to a big problem, since the CLBs where the only elements where the Pof was no possible to trace down to individual instances or signal within the design. This in turn lead to the design where everything was triplicated, the device TMR, to be the most robust. While all the other designs, even when in some cases tolerating all faults in the FFs and BRAMs, lacked behind due to the mitigation in the CLBs not being sufficient. In order to achieve better better values for the FITs more tests on designs targeting the CLBs should have been carried out, using the results as a means of tracking down the parts of the design mapped into CLBs that were most vulnerable. Applying TMR to these parts of the design would have drastically reduced the contribution of the CLBs to the FITs, as could be seen when triplicating the ALU for the first designs.

The calculations for each of the designs, their consumed resources and their chip floor-plan, can all be seen in the appendix.

Design (4 FAULT)	EBs Used				Chip Pow. Cons. (W)	Max Op. Freq. (MHz)	PoF (%)				FIT / RoF (FIT/Mb)			
	CLB	BRAMs	FFs	Total			CLB	BRAMs	FFs	Total	CLB	BRAMs	FFs	Total
Device TMR	794569	221184	1281	1017034	1.57	50	3.55 ± 0.08	1.95 ± 0.12	0.007	5.51 ± 0.20	3.615 ± 0.074	1.986 ± 0.124	0.007	5.609 ± 0.203
TMR & EDAC	667775	221184	812	889771	1.57	50	9.45 ± 0.15	0	0.002	2.55 ± 0.08	8.405 ± 0.134	0	0.001	8.406 ± 0.134
Device TMR and EDAC	1962163	663552	2361	2628076	1.63	50	3.34 ± 0.07	0	0	3.34 ± 0.07	8.779 ± 0.196	0	0	8.779 ± 0.196

Figure 4.7: Comparison of Resources, PoFs and FITs for the Different Designs (4 Faults).

CHAPTER 5

Discussion

The results presented in the previous section deal only with the fault tolerant aspect of device dependability. Yet in any normal space design these results would consist of nothing else but the initial step in determining the arrangement of techniques to be further deployed. When having the FITs, any professional designer would proceed to design a fault removal model based on scrubbing in order to avoid the accumulation of faults. The reason being that the FITs represent soft errors that will permanently reconfigure the FPGA's configuration memory. These soft errors can be mitigated by the fault tolerant techniques for a limited amount of time before the failure rate of the device reaches unacceptable values as the SEUs accumulate. This is an extremely important aspect of any space system and even though it was not implemented in the designs presented, it is a clear next step for future projects.

Another crucial aspect of space radiation in electronics is that of the TID or very high energy particles, both of which will cause hard errors in the FPGA. These kind of errors pose the same threat to the design's failure rate as soft errors do with the risk of accumulation. Yet these errors cannot be fixed with scrubbing, they can only be ignored by reallocating the design. Once again any professional designer would have to take into account this when assessing the device's dependability and despite no mechanism like this being implemented in the solutions presented, it is a very important aspect to keep in mind for future, more complex iterations of this project.

Time is a deciding element for any space system. It is the force against which device dependability struggles the most and it is inevitable. The unit of FIT is based on time for this very reason, cause no matter how robust a design is it will fail at some point. For instance a device with a failure rate of 1 FIT is a very robust device as in a matter of 1000 million hours it is estimated to fail once. Yet, even if the amount of time till failure is as high as 1000 million hours, it still is a failure in time. This is why it is important to determine the aspect of space system design known as mission duration, a measure of the time required for a mission to carry out its intended purpose. This is an extremely important aspect of space system design that was not discussed throughout the thesis and it is required in order to asses whether the obtained FITs are acceptable or not. For example if a system is intended to execute an application for 15000 years, a failure rate of 1 FIT would not be acceptable as it would entail that the system would fail once during its functioning. Another important aspect required to conclude whether a failure rate is acceptable or not is the importance of the system, something that normally is decided based on

human life, with systems that are crucial to human survival as the most critical. A combination of mission duration and system importance is what would be used to set the dependability margins for the space system to be designed, yet in the thesis this elements are not discussed due to the lack of a specific space application.

On a final note, this thesis was an extremely interesting project that allowed me to establish a basis on a topic of which I new very little before starting. This also meant that a lot of time was employed on just getting up to date on the theory behind space radiation in FPGAs and the state of the art fault tolerant techniques. Being an electronics student, the areas of space physics and space weather were completely new to me. This in turned limited my understanding of these topics to a more or less superficial level. For future revisions of this project a better theoretical background would be preferred and as such I will take several specialization courses on these topics for my masters. In a way I wish I could start it all over from the point I am currently standing at, in order to present a more concise and structure thesis. After all, when starting this thesis I though I was aiming to create a fault tolerant system with a specific purpose. It was not until the final days of the project that I realised that there was no real concise objective present in the work. The AVR was made fault tolerant to arbitrary numbers of faults, so rather than becoming a true example of what would be a real life fault tolerant system it became a collection of techniques I applied in order to get the ropes of the topic and improve failure rates in an all-together non realistic way. Yet I do not feel like the effort to understand the topic was a waste, since now I have the certainty that this is the field of hardware design I want to pursue. I plan to delve deeper into space-level hardware dependability for my master's thesis, where I hopefully will be able to implement the more advanced techniques and concepts commented in this discussion that only now I am starting to understand, while doing so in a realistic and practical manner.

APPENDIX A

Calculations, Resources and Floor-Plans

Basic AVR

$EB_{CLB} := 279850$:

$EB_{RAM} := 73728$:

$EB_{FF} := 460$:

$EB_{DUT} := EB_{CLB} + EB_{RAM} + EB_{FF}$

$$EB_{DUT} := 354038 \tag{1.1.1}$$

$$PoF_{CLB} := \left(\frac{10.6 \cdot EB_{CLB}}{EB_{DUT}}\right) \pm \left(\frac{0.1 \cdot EB_{CLB}}{EB_{DUT}}\right)$$

$$PoF_{CLB} := 8.378789847 \pm 0.07904518724 \tag{1.1.2}$$

$$PoF_{RAM} := \left(\frac{3.88 \cdot EB_{RAM}}{EB_{DUT}}\right) \pm \left(\frac{0.54 \cdot EB_{RAM}}{EB_{DUT}}\right)$$

$$PoF_{RAM} := 0.8080054684 \pm 0.1124543693 \tag{1.1.3}$$

$$PoF_{FF} := \left(\frac{14.08\, EB_{FF}}{EB_{DUT}}\right) \pm \left(\frac{0.98 \cdot EB_{FF}}{EB_{DUT}}\right)$$

$$PoF_{FF} := 0.01829408143 \pm 0.001273309645 \tag{1.1.4}$$

$$PoF_{DUT} := (8.378789847 + 0.8080054684 + 0.01829408143) \pm (0.07904518724 + 0.1124543693 + 0.001273309645)$$

$$9.21 \pm 0.19 \tag{1.1.5}$$

$$FIT_{CLB}\left(\frac{FIT}{Mb}\right) := \left(\left(10.6 \cdot \frac{EB_{CLB}}{1000000}\right) \pm \left(0.1 \cdot \frac{EB_{CLB}}{1000000}\right)\right) \cdot FR$$

$$FIT_{CLB}\left(\frac{FIT}{Mb}\right) := (2.966410000 \pm 0.02798500000)\, FR \tag{1.1.6}$$

$$FIT_{RAM}\left(\frac{FIT}{Mb}\right) := \left(\left(3.88 \cdot \frac{EB_{RAM}}{1000000}\right) \pm \left(0.54 \cdot \frac{EB_{RAM}}{1000000}\right)\right) FR$$

$$FIT_{RAM}\left(\frac{FIT}{Mb}\right) := (0.2860646400 \pm 0.03981312000)\, FR \tag{1.1.7}$$

$$FIT_{FF}\left(\frac{FIT}{Mb}\right) := \left(\left(14.08 \cdot \frac{EB_{FF}}{1000000}\right) \pm \left(0.98 \cdot \frac{EB_{FF}}{1000000}\right)\right) FR$$

$$FIT_{FF}\left(\frac{FIT}{Mb}\right) := (0.006476800000 \pm 0.0004508000000)\, FR \tag{1.1.8}$$

$$FIT_{DUT}\left(\frac{FIT}{Mb}\right) := \left(\left(9.205089396 \cdot \frac{EB_{DUT}}{1000000}\right) \pm \left(0.1927728661 \cdot \frac{EB_{DUT}}{1000000}\right)\right) \cdot FR$$

$$FR_{DUT}\left(\frac{FIT}{Mb}\right) := 3.258951440 \pm 0.06824891997 \tag{1.1.9}$$

Figure A.1: Calculations for the Basic AVR Design.

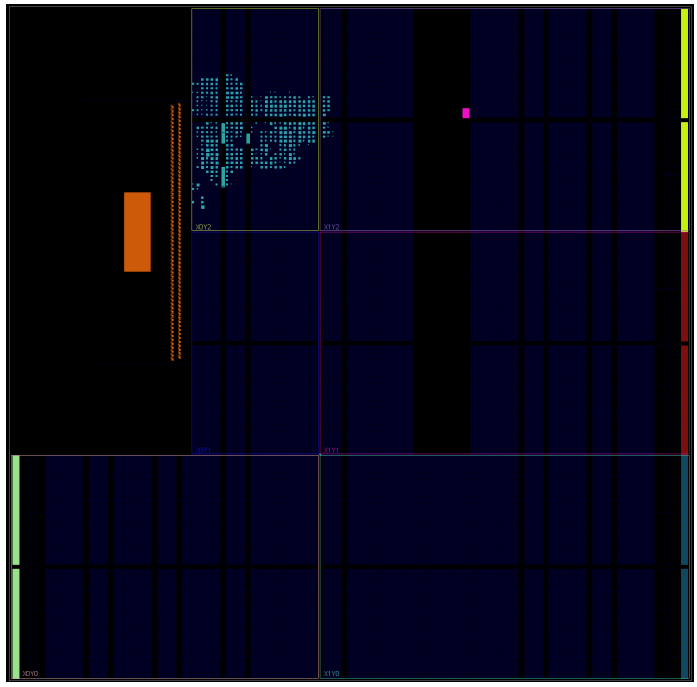

Figure A.2: Floor-Plan for the Basic AVR Design.

Resource	Utilization	Available	Utilization...
LUT	1322	53200	2.48
LUTRAM	1	17400	0.01
FF	460	106400	0.43
BRAM	2	140	1.43
DSP	1	220	0.45
BUFG	1	32	3.13

Figure A.3: Resources employed by the Basic AVR Design.

Device TMR

$EB_{CLB} := 794569$:

$EB_{RAM} := 221184$:

$EB_{FF} := 1281$:

$EB_{DUT} := EB_{CLB} + EB_{RAM} + EB_{FF}$

$$EB_{DUT} := 1017034 \tag{1.2.1}$$

$$PoF_{CLB} := \left(\frac{0 \cdot EB_{CLB}}{EB_{DUT}}\right) \pm \left(\frac{0 \cdot EB_{CLB}}{EB_{DUT}}\right)$$

$$PoF_{CLB} := \pm 0 \tag{1.2.2}$$

$$PoF_{RAM} := \left(\frac{0 \cdot EB_{RAM}}{EB_{DUT}}\right) \pm \left(\frac{0 \cdot EB_{RAM}}{EB_{DUT}}\right)$$

$$PoF_{RAM} := \pm 0 \tag{1.2.3}$$

$$PoF_{FF} := \left(\frac{0\ EB_{FF}}{EB_{DUT}}\right) \pm \left(\frac{0 \cdot EB_{FF}}{EB_{DUT}}\right)$$

$$PoF_{FF} := \pm 0 \tag{1.2.4}$$

$PoF_{DUT} := (0 + 0 + 0) \pm (0 + 0 + 0)$

$$\pm 0.00 \tag{1.2.5}$$

$$FIT_{CLB}\left(\frac{FIT}{Mb}\right) := \left(\left(0 \cdot \frac{EB_{CLB}}{1000000}\right) \pm \left(0 \cdot \frac{EB_{CLB}}{1000000}\right)\right) \cdot FR$$

$$FIT_{CLB}\left(\frac{FIT}{Mb}\right) := (\pm 0)\ FR \tag{1.2.6}$$

$$FIT_{RAM}\left(\frac{FIT}{Mb}\right) := \left(\left(0 \cdot \frac{EB_{RAM}}{1000000}\right) \pm \left(0 \cdot \frac{EB_{RAM}}{1000000}\right)\right) FR$$

$$FIT_{RAM}\left(\frac{FIT}{Mb}\right) := (\pm 0)\ FR \tag{1.2.7}$$

$$FIT_{FF}\left(\frac{FIT}{Mb}\right) := \left(\left(0 \cdot \frac{EB_{FF}}{1000000}\right) \pm \left(0 \cdot \frac{EB_{FF}}{1000000}\right)\right) FR$$

$$FIT_{FF}\left(\frac{FIT}{Mb}\right) := (\pm 0)\ FR \tag{1.2.8}$$

$$FIT_{DUT}\left(\frac{FIT}{Mb}\right) := \left(\left(0 \cdot \frac{EB_{DUT}}{1000000}\right) \pm \left(0 \cdot \frac{EB_{DUT}}{1000000}\right)\right) \cdot FR$$

$$FIT_{DUT}\left(\frac{FIT}{Mb}\right) := (\pm 0)\ FR \tag{1.2.9}$$

Figure A.4: Calculations for the Device TMR Design.

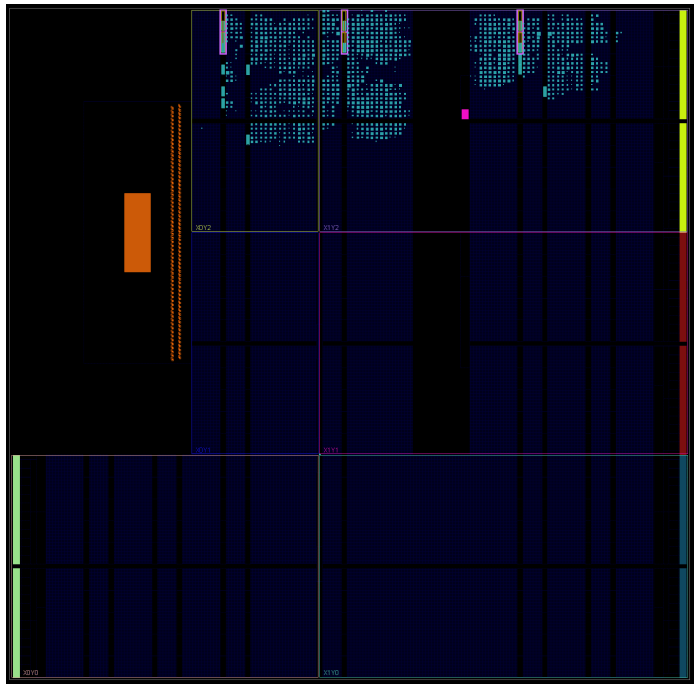

Figure A.5: Floor-Plan for the Device TMR Design.

Resource	Utilization	Available	Utilization...
LUT	3951	53200	7.43
LUTRAM	1	17400	0.01
FF	1281	106400	1.20
BRAM	6	140	4.29
DSP	3	220	1.36
BUFG	1	32	3.13

Figure A.6: Resources employed by the Device TMR Design.

TMR ALU

$EB_{CLB1} := 447502$:

$EB_{CLB3} := 450024$:

$EB_{RAM} := 73728$:

$EB_{FF} := 460$:

$EB_{DUT1} := EB_{CLB1} + EB_{RAM} + EB_{FF}$

$$EB_{DUT1} := 521690 \quad (1.4.1)$$

$EB_{DUT3} := EB_{CLB3} + EB_{RAM} + EB_{FF}$

$$EB_{DUT3} := 524212 \quad (1.4.2)$$

$$PoF_{CLB1} := \left(\frac{5.483 \cdot EB_{CLB1}}{EB_{DUT1}}\right) \pm \left(\frac{0.1 \cdot EB_{CLB1}}{EB_{DUT1}}\right)$$

$$PoF_{CLB1} := 4.703278702 \pm 0.08577929422 \quad (1.4.3)$$

$$PoF_{CLB3} := \left(\frac{5.472 \cdot EB_{CLB3}}{EB_{DUT3}}\right) \pm \left(\frac{0.1 \cdot EB_{CLB3}}{EB_{DUT3}}\right)$$

$$PoF_{CLB3} := 4.697586717 \pm 0.08584771047 \quad (1.4.4)$$

$$PoF_{RAM} := \left(\frac{3.88 \cdot EB_{RAM}}{EB_{DUT1}}\right) \pm \left(\frac{0.54 \cdot EB_{RAM}}{EB_{DUT1}}\right)$$

$$PoF_{RAM} := 0.5483421955 \pm 0.07631566639 \quad (1.4.5)$$

$$PoF_{FF} := \left(\frac{14.08\, EB_{FF}}{EB_{DUT1}}\right) \pm \left(\frac{0.98 \cdot EB_{FF}}{EB_{DUT1}}\right)$$

$$PoF_{FF} := 0.01241503575 \pm 0.0008641147041 \quad (1.4.6)$$

$$PoF_{DUT} := (4.703278702 + 0.5483421955 + 0.01241503575) \pm (0.08577929422 + 0.07631566639 + 0.0008641147041)$$

$$5.26 \pm 0.16 \quad (1.4.7)$$

$$FIT_{CLB1}\left(\frac{FIT}{Mb}\right) := \left(\left(5.483 \cdot \frac{EB_{CLB1}}{1000000}\right) \pm \left(0.1 \cdot \frac{EB_{CLB1}}{1000000}\right)\right) \cdot FR$$

$$FIT_{CLB1}\left(\frac{FIT}{Mb}\right) := (2.453653466 \pm 0.04475020000)\ FR \quad (1.4.8)$$

$$FIT_{CLB3}\left(\frac{FIT}{Mb}\right) := \left(\left(5.472 \cdot \frac{EB_{CLB3}}{1000000}\right) \pm \left(0.1 \cdot \frac{EB_{CLB3}}{1000000}\right)\right) \cdot FR$$

$$FIT_{CLB3}\left(\frac{FIT}{Mb}\right) := (2.462531328 \pm 0.04500240000)\ FR \quad (1.4.9)$$

$$FIT_{RAM}\left(\frac{FIT}{Mb}\right) := \left(\left(3.88 \cdot \frac{EB_{RAM}}{1000000}\right) \pm \left(0.54 \cdot \frac{EB_{RAM}}{1000000}\right)\right) FR$$

$$FIT_{RAM}\left(\frac{FIT}{Mb}\right) := (0.2860646400 \pm 0.03981312000)\ FR \quad (1.4.10)$$

$$FIT_{FF}\left(\frac{FIT}{Mb}\right) := \left(\left(14.08 \cdot \frac{EB_{FF}}{1000000}\right) \pm \left(0.98 \cdot \frac{EB_{FF}}{1000000}\right)\right) FR$$

$$FIT_{FF}\left(\frac{FIT}{Mb}\right) := (0.006476800000 \pm 0.0004508000000)\ FR \quad (1.4.11)$$

$$FIT_{DUT}\left(\frac{FIT}{Mb}\right) := \left(\left(5.264035934 \cdot \frac{EB_{DUT1}}{1000000}\right) \pm \left(0.1629590753 \cdot \frac{EB_{DUT1}}{1000000}\right)\right) \cdot FR$$

$$FIT_{DUT}\left(\frac{FIT}{Mb}\right) := (2.746194906 \pm 0.08501411999)\ FR \quad (1.4.12)$$

Figure A.7: Calculations for the ALU TMR Design.

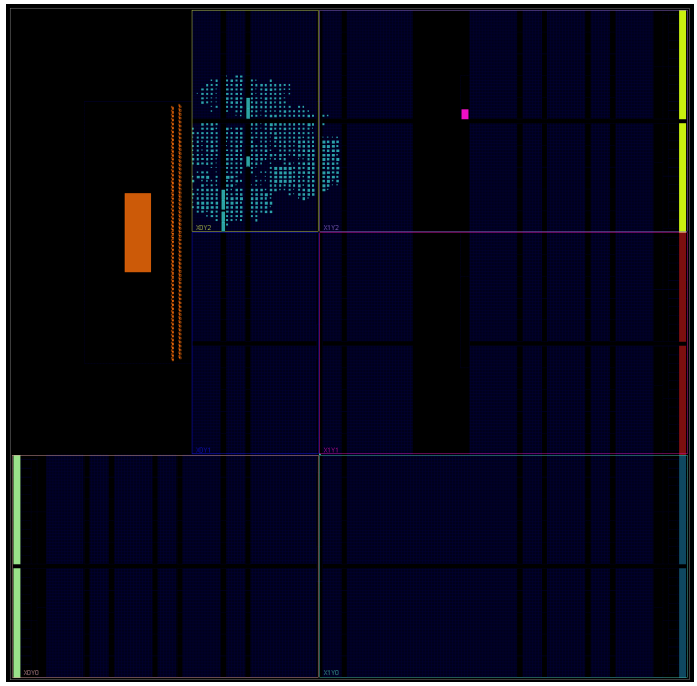

Figure A.8: Floor-Plan for the ALU TMR Design.

Resource	Utilization	Available	Utilizatio...
LUT	2153	53200	4.05
LUTRAM	1	17400	0.01
FF	455	106400	0.43
BRAM	2	140	1.43
DSP	3	220	1.36
BUFG	1	32	3.13

Figure A.9: Resources employed by the ALU TMR Design.

TMR DECO

$EB_{CLB} := 392713$:

$EB_{RAM} := 73728$:

$EB_{FF} := 659$:

$EB_{DUT} := EB_{CLB} + EB_{RAM} + EB_{FF}$

$$EB_{DUT} := 467100 \tag{1.5.1}$$

$$PoF_{CLB} := \left(\frac{7.467 \cdot EB_{CLB}}{EB_{DUT}}\right) \pm \left(\frac{0.1 \cdot EB_{CLB}}{EB_{DUT}}\right)$$

$$PoF_{CLB} := 6.277859069 \pm 0.08407471634 \tag{1.5.2}$$

$$PoF_{RAM} := \left(\frac{3.88 \cdot EB_{RAM}}{EB_{DUT}}\right) \pm \left(\frac{0.54 \cdot EB_{RAM}}{EB_{DUT}}\right)$$

$$PoF_{RAM} := 0.6124269749 \pm 0.08523468208 \tag{1.5.3}$$

$$PoF_{FF} := \left(\frac{11.22\, EB_{FF}}{EB_{DUT}}\right) \pm \left(\frac{0.98 \cdot EB_{FF}}{EB_{DUT}}\right)$$

$$PoF_{FF} := 0.01582954399 \pm 0.001382616142 \tag{1.5.4}$$

$$PoF_{DUT} := (6.277859069 + 0.6124269749 + 0.01582954399) \pm (0.08407471634 + 0.08523468208 + 0.001382616142)$$

$$6.91 \pm 0.17 \tag{1.5.5}$$

$$FIT_{CLB}\left(\frac{FIT}{Mb}\right) := \left(\left(7.467 \cdot \frac{EB_{CLB}}{1000000}\right) \pm \left(0.1 \cdot \frac{EB_{CLB}}{1000000}\right)\right) \cdot FR$$

$$FIT_{CLB}\left(\frac{FIT}{Mb}\right) := (2.932387971 \pm 0.03927130000)\, FR \tag{1.5.6}$$

$$FIT_{RAM}\left(\frac{FIT}{Mb}\right) := \left(\left(3.88 \cdot \frac{EB_{RAM}}{1000000}\right) \pm \left(0.54 \cdot \frac{EB_{RAM}}{1000000}\right)\right) FR$$

$$FIT_{RAM}\left(\frac{FIT}{Mb}\right) := (0.2860646400 \pm 0.03981312000)\, FR \tag{1.5.7}$$

$$FIT_{FF}\left(\frac{FIT}{Mb}\right) := \left(\left(11.22 \cdot \frac{EB_{FF}}{1000000}\right) \pm \left(0.98 \cdot \frac{EB_{FF}}{1000000}\right)\right) FR$$

$$FIT_{FF}\left(\frac{FIT}{Mb}\right) := (0.007393980000 \pm 0.0006458200000)\, FR \tag{1.5.8}$$

$$FIT_{DUT}\left(\frac{FIT}{Mb}\right) := \left(\left(6.906115588 \cdot \frac{EB_{DUT}}{1000000}\right) \pm \left(0.1706920145 \cdot \frac{EB_{DUT}}{1000000}\right)\right) \cdot FR$$

$$FIT_{DUT}\left(\frac{FIT}{Mb}\right) := (3.225846591 \pm 0.07973023997)\, FR \tag{1.5.9}$$

Figure A.10: Calculations for the OPC-DECO TMR Design.

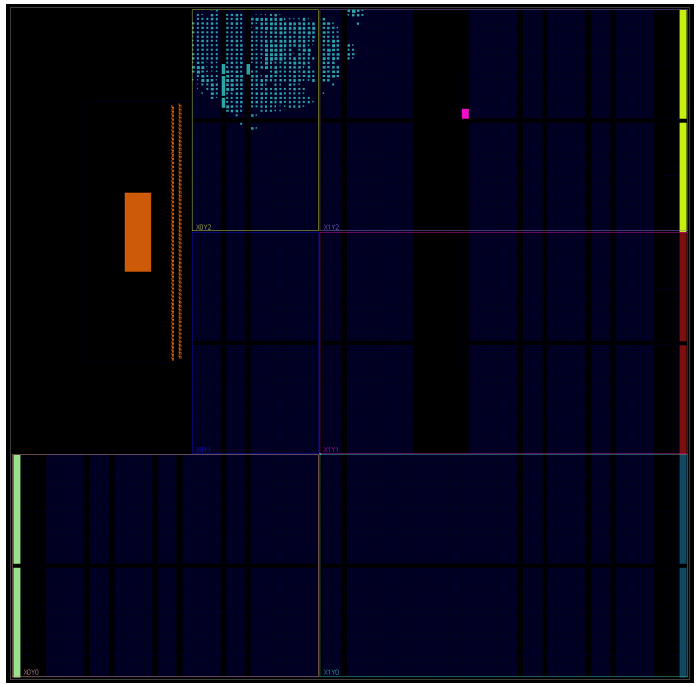

Figure A.11: Floor-Plan for the OPC-DECO TMR Design.

Resource	Utilization	Available	Utilization %
LUT	1955	53200	3.67
LUTRAM	1	17400	0.01
FF	659	106400	0.62
BRAM	2	140	1.43
DSP	1	220	0.45
BUFG	1	32	3.13

Figure A.12: Resources employed by the OPC-DECO TMR Design.

EDAC

$EB_{CLB} := 405615$:

$EB_{RAM} := 221184$:

$EB_{FF} := 606$:

$EB_{DUT} := EB_{CLB} + EB_{RAM} + EB_{FF}$

$$EB_{DUT} := 627405 \tag{1.3.1}$$

$$PoF_{CLB} := \left(\frac{8.465 \cdot EB_{CLB}}{EB_{DUT}}\right) \pm \left(\frac{0.1 \cdot EB_{CLB}}{EB_{DUT}}\right)$$

$$PoF_{CLB} := 5.472591030 \pm 0.06464962823 \tag{1.3.2}$$

$$PoF_{RAM} := \left(\frac{0 \cdot EB_{RAM}}{EB_{DUT}}\right) \pm \left(\frac{0 \cdot EB_{RAM}}{EB_{DUT}}\right)$$

$$PoF_{RAM} := \pm 0 \tag{1.3.3}$$

$$PoF_{FF} := \left(\frac{2.3\ EB_{FF}}{EB_{DUT}}\right) \pm \left(\frac{0.93 \cdot EB_{FF}}{EB_{DUT}}\right)$$

$$PoF_{FF} := 0.002221531547 \pm 0.0008982714515 \tag{1.3.4}$$

$$PoF_{DUT} := (5.472591030 + 0 + 0.002221531547) \pm (0.06464962823 + 0 + 0.0008982714515)$$

$$5.47 \pm 0.07 \tag{1.3.5}$$

$$FIT_{CLB}\left(\frac{FIT}{Mb}\right) := \left(\left(8.465 \cdot \frac{EB_{CLB}}{1000000}\right) \pm \left(0.1 \cdot \frac{EB_{CLB}}{1000000}\right)\right) \cdot FR$$

$$FIT_{CLB}\left(\frac{FIT}{Mb}\right) := (3.433530975 \pm 0.04056150000)\ FR \tag{1.3.6}$$

$$FIT_{RAM}\left(\frac{FIT}{Mb}\right) := \left(\left(0 \cdot \frac{EB_{RAM}}{1000000}\right) \pm \left(0 \cdot \frac{EB_{RAM}}{1000000}\right)\right) FR$$

$$FIT_{RAM}\left(\frac{FIT}{Mb}\right) := (\pm 0)\ FR \tag{1.3.7}$$

$$FIT_{FF}\left(\frac{FIT}{Mb}\right) := \left(\left(2.3 \cdot \frac{EB_{FF}}{1000000}\right) \pm \left(0.93 \cdot \frac{EB_{FF}}{1000000}\right)\right) FR$$

$$FIT_{FF}\left(\frac{FIT}{Mb}\right) := (0.001867600000 \pm 0.0007551600000)\ FR \tag{1.3.8}$$

$$FIT_{DUT}\left(\frac{FIT}{Mb}\right) := \left(\left(5.474812562 \cdot \frac{EB_{DUT}}{1000000}\right) \pm \left(0.06554789968 \cdot \frac{EB_{DUT}}{1000000}\right)\right) \cdot FR$$

$$FIT_{DUT}\left(\frac{FIT}{Mb}\right) := (3.434924775 \pm 0.04112508000)\ FR \tag{1.3.9}$$

Figure A.13: Calculations for the EDAC in BRAMs and FFs Design.

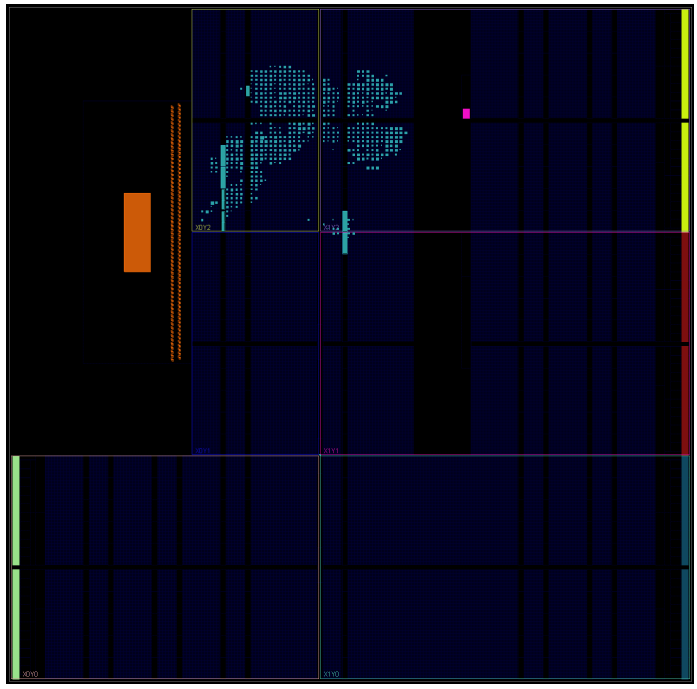

Figure A.14: Floor-Plan for the EDAC in BRAMs and FFs Design.

Resource	Utilization	Available	Utilization %
LUT	2027	53200	3.81
LUTRAM	1	17400	0.01
FF	606	106400	0.57
BRAM	6	140	4.29
DSP	1	220	0.45
BUFG	1	32	3.13

Figure A.15: Resources employed by the EDAC in BRAMs and FFs Design.

TMR and EDAC

$EB_{CLB} := 667775$:

$EB_{RAM} := 221184$:

$EB_{FF} := 812$:

$EB_{DUT} := EB_{CLB} + EB_{RAM} + EB_{FF}$

$$EB_{DUT} := 889771 \tag{1.6.1}$$

$$PoF_{CLB} := \left(\frac{3.4 \cdot EB_{CLB}}{EB_{DUT}}\right) \pm \left(\frac{0.1 \cdot EB_{CLB}}{EB_{DUT}}\right)$$

$$PoF_{CLB} := 2.551707125 \pm 0.07505020955 \tag{1.6.2}$$

$$PoF_{RAM} := \left(\frac{0 \cdot EB_{RAM}}{EB_{DUT}}\right) \pm \left(\frac{0 \cdot EB_{RAM}}{EB_{DUT}}\right)$$

$$PoF_{RAM} := \pm 0 \tag{1.6.3}$$

$$PoF_{FF} := \left(\frac{0.5\, EB_{FF}}{EB_{DUT}}\right) \pm \left(\frac{0.93 \cdot EB_{FF}}{EB_{DUT}}\right)$$

$$PoF_{FF} := 0.0004562971821 \pm 0.0008487127587 \tag{1.6.4}$$

$$PoF_{DUT} := (2.551707125 + 0 + 0.0004562971821) \pm (0.07505020955 + 0 + 0.0008487127587)$$

$$2.55 \pm 0.08 \tag{1.6.5}$$

$$FIT_{CLB}\left(\frac{FIT}{Mb}\right) := \left(\left(3.4 \cdot \frac{EB_{CLB}}{1000000}\right) \pm \left(0.1 \cdot \frac{EB_{CLB}}{1000000}\right)\right) \cdot FR$$

$$FIT_{CLB}\left(\frac{FIT}{Mb}\right) := (2.270435000 \pm 0.06677750000)\, FR \tag{1.6.6}$$

$$FIT_{RAM}\left(\frac{FIT}{Mb}\right) := \left(\left(0 \cdot \frac{EB_{RAM}}{1000000}\right) \pm \left(0 \cdot \frac{EB_{RAM}}{1000000}\right)\right) FR$$

$$FIT_{RAM}\left(\frac{FIT}{Mb}\right) := (\pm 0)\, FR \tag{1.6.7}$$

$$FIT_{FF}\left(\frac{FIT}{Mb}\right) := \left(\left(0.5 \cdot \frac{EB_{FF}}{1000000}\right) \pm \left(0.93 \cdot \frac{EB_{FF}}{1000000}\right)\right) FR$$

$$FIT_{FF}\left(\frac{FIT}{Mb}\right) := (0.0004060000000 \pm 0.0007551600000)\, FR \tag{1.6.8}$$

$$FIT_{DUT}\left(\frac{FIT}{Mb}\right) := \left(\left(2.552163422 \cdot \frac{EB_{DUT}}{1000000}\right) \pm \left(0.07589892231 \cdot \frac{EB_{DUT}}{1000000}\right)\right) \cdot FR$$

$$FIT_{DUT}\left(\frac{FIT}{Mb}\right) := (2.270841000 \pm 0.06753266000)\, FR \tag{1.6.9}$$

Figure A.16: Calculations for the Design with TMRs and EDAC.

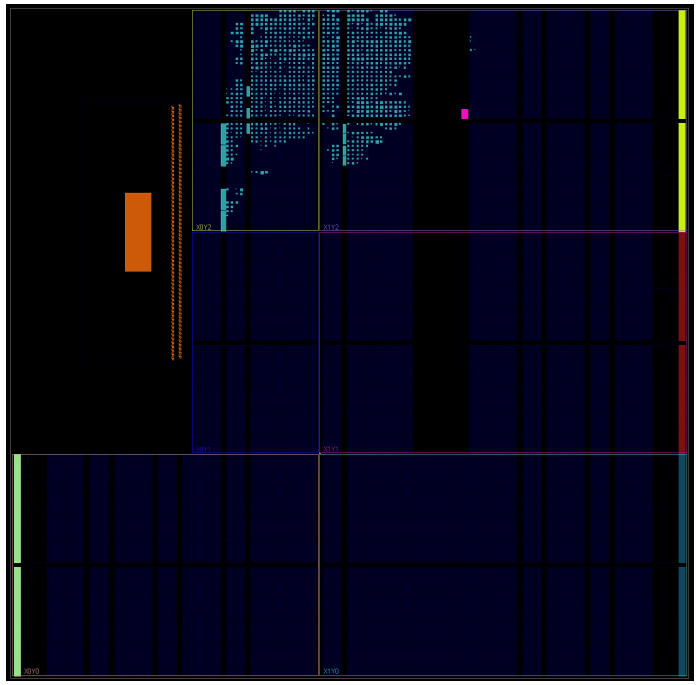

Figure A.17: Floor-Plan for the Design with TMRs and EDAC.

Resource	Utilization	Available	Utilization %
LUT	3415	53200	6.42
LUTRAM	1	17400	0.01
FF	812	106400	0.76
BRAM	6	140	4.29
DSP	3	220	1.36
BUFG	1	32	3.13

Figure A.18: Resources employed by the Design with TMRs and EDAC.

Device TMR

$EB_{CLB} := 794569$:

$EB_{RAM} := 221184$:

$EB_{FF} := 1281$:

$EB_{DUT} := EB_{CLB} + EB_{RAM} + EB_{FF}$

$$EB_{DUT} := 1017034 \tag{2.1.1}$$

$$FoF_{CLB} := \left(\frac{4.55 \cdot EB_{CLB}}{EB_{TOT}}\right) \pm \left(\frac{0.1 \cdot EB_{CLB}}{EB_{TOT}}\right)$$

$$FoF_{CLB} := 3.554737551 \pm 0.07812610001 \tag{2.1.2}$$

$$FoF_{RAM} := \left(\frac{8.98 \cdot EB_{RAM}}{EB_{TOT}}\right) \pm \left(\frac{0.56 \cdot EB_{RAM}}{EB_{TOT}}\right)$$

$$FoF_{RAM} := 1.952965506 \pm 0.1217884948 \tag{2.1.3}$$

$$FoF_{FF} := \left(\frac{5.52\, EB_{FF}}{EB_{TOT}}\right) \pm \left(\frac{0.09 \cdot EB_{FF}}{EB_{TOT}}\right)$$

$$FoF_{FF} := 0.006952687914 \pm 0.0001133590421 \tag{2.1.4}$$

$$FoF_{TOT} := (3.554737551 + 1.952965506 + 0.006952687914) \pm (0.07812610001 + 0.1217884948 + 0.0001133590421)$$

$$5.51 \pm 0.20 \tag{2.1.5}$$

$$FIT_{CLB}\left(\frac{FIT}{Mb}\right) := \left(\left(4.55 \cdot \frac{EB_{CLB}}{1000000}\right) \pm \left(0.1 \cdot \frac{EB_{CLB}}{1000000}\right)\right) \cdot FR$$

$$FIT_{CLB}\left(\frac{FIT}{Mb}\right) := (3.615288950 \pm 0.07945690000)\, FR \tag{2.1.6}$$

$$FIT_{RAM}\left(\frac{FIT}{Mb}\right) := \left(\left(8.98 \cdot \frac{EB_{RAM}}{1000000}\right) \pm \left(0.56 \cdot \frac{EB_{RAM}}{1000000}\right)\right) FR$$

$$FIT_{RAM}\left(\frac{FIT}{Mb}\right) := (1.986232320 \pm 0.1238630400)\, FR \tag{2.1.7}$$

$$FIT_{FF}\left(\frac{FIT}{Mb}\right) := \left(\left(5.52 \cdot \frac{EB_{FF}}{1000000}\right) \pm \left(0.09 \cdot \frac{EB_{FF}}{1000000}\right)\right) FR$$

$$FIT_{FF}\left(\frac{FIT}{Mb}\right) := (0.007071120000 \pm 0.0001152900000)\, FR \tag{2.1.8}$$

$$FIT_{DUT}\left(\frac{FIT}{Mb}\right) := \left(\left(5.514655745 \cdot \frac{EB_{DUT}}{1000000}\right) \pm \left(0.2000279538 \cdot \frac{EB_{DUT}}{1000000}\right)\right) \cdot FR$$

$$FIT_{DUT}\left(\frac{FIT}{Mb}\right) := (5.608592391 \pm 0.2034352300)\, FR \tag{2.1.9}$$

Figure A.19: Calculations for the Device TMR, 4 Faults.

TMR and EDAC

$EB_{CLB} := 667775$:

$EB_{RAM} := 221184$:

$EB_{FF} := 812$:

$EB_{DUT} := EB_{CLB} + EB_{RAM} + EB_{FF}$

$$EB_{DUT} := 889771 \tag{2.2.1}$$

$$PoF_{CLB} := \left(\frac{12.586 \cdot EB_{CLB}}{EB_{DUT}}\right) \pm \left(\frac{0.2 \cdot EB_{CLB}}{EB_{DUT}}\right)$$

$$PoF_{CLB} := 9.445819374 \pm 0.1501004191 \tag{2.2.2}$$

$$PoF_{RAM} := \left(\frac{0 \cdot EB_{RAM}}{EB_{DUT}}\right) \pm \left(\frac{0 \cdot EB_{RAM}}{EB_{DUT}}\right)$$

$$PoF_{RAM} := \pm 0 \tag{2.2.3}$$

$$PoF_{FF} := \left(\frac{1.96 \cdot EB_{FF}}{EB_{DUT}}\right) \pm \left(\frac{0.54 \cdot EB_{FF}}{EB_{DUT}}\right)$$

$$PoF_{FF} := 0.001788684954 \pm 0.0004928009567 \tag{2.2.4}$$

$$PoF_{DUT} := (9.445819374 + 0 + 0.001788684954) \pm (0.1501004191 + 0 + 0.0004928009567)$$

$$9.45 \pm 0.15 \tag{2.2.5}$$

$$FIT_{CLB}\left(\frac{FIT}{Mb}\right) := \left(\left(12.586 \cdot \frac{EB_{CLB}}{1000000}\right) \pm \left(0.2 \cdot \frac{EB_{CLB}}{1000000}\right)\right) \cdot FR$$

$$FIT_{CLB}\left(\frac{FIT}{Mb}\right) := (8.404616150 \pm 0.1335550000)\ FR \tag{2.2.6}$$

$$FIT_{RAM}\left(\frac{FIT}{Mb}\right) := \left(\left(0 \cdot \frac{EB_{RAM}}{1000000}\right) \pm \left(0 \cdot \frac{EB_{RAM}}{1000000}\right)\right) FR$$

$$FIT_{RAM}\left(\frac{FIT}{Mb}\right) := (\pm 0)\ FR \tag{2.2.7}$$

$$FIT_{FF}\left(\frac{FIT}{Mb}\right) := \left(\left(1.96 \cdot \frac{EB_{FF}}{1000000}\right) \pm \left(0.54 \cdot \frac{EB_{FF}}{1000000}\right)\right) FR$$

$$FIT_{FF}\left(\frac{FIT}{Mb}\right) := (0.001591520000 \pm 0.0004384800000)\ FR \tag{2.2.8}$$

$$FIT_{DUT}\left(\frac{FIT}{Mb}\right) := \left(\left(9.447608059 \cdot \frac{EB_{DUT}}{1000000}\right) \pm \left(0.1505932201 \cdot \frac{EB_{DUT}}{1000000}\right)\right) \cdot FR$$

$$FIT_{DUT}\left(\frac{FIT}{Mb}\right) := (8.406207670 \pm 0.1339934800)\ FR \tag{2.2.9}$$

Figure A.20: Calculations for the Design with TMRs and EDAC, 4 Faults.

DEVICE , TMR and EDAC

$EB_{CLB} := 1962163$:

$EB_{RAM} := 663552$:

$EB_{FF} := 2361$:

$EB_{DUT} := EB_{CLB} + EB_{RAM} + EB_{FF}$

$$EB_{DUT} := 2628076 \tag{2.3.1}$$

$$PoF_{CLB} := \left(\frac{4.474 \cdot EB_{CLB}}{EB_{DUT}}\right) \pm \left(\frac{0.1 \cdot EB_{CLB}}{EB_{DUT}}\right)$$

$$PoF_{CLB} := 3.340358978 \pm 0.07466157752 \tag{2.3.2}$$

$$PoF_{RAM} := \left(\frac{0 \cdot EB_{RAM}}{EB_{DUT}}\right) \pm \left(\frac{0 \cdot EB_{RAM}}{EB_{DUT}}\right)$$

$$PoF_{RAM} := \pm 0 \tag{2.3.3}$$

$$PoF_{FF} := \left(\frac{1.96 \cdot EB_{FF}}{EB_{DUT}}\right) \pm \left(\frac{0.54 \cdot EB_{FF}}{EB_{DUT}}\right)$$

$$PoF_{FF} := 0.001788684954 \pm 0.0004928009567 \tag{2.3.4}$$

$$PoF_{DUT} := (3.340358978 + 0 + 0.001788684954) \pm (0.07466157752 + 0 + 0.0004928009567)$$

$$3.34 \pm 0.08 \tag{2.3.5}$$

$$FIT_{CLB}\left(\frac{FIT}{Mb}\right) := \left(\left(4.474 \cdot \frac{EB_{CLB}}{1000000}\right) \pm \left(0.1 \cdot \frac{EB_{CLB}}{1000000}\right)\right) \cdot FR$$

$$FIT_{CLB}\left(\frac{FIT}{Mb}\right) := (8.778717262 \pm 0.1962163000)\ FR \tag{2.3.6}$$

$$FIT_{RAM}\left(\frac{FIT}{Mb}\right) := \left(\left(0 \cdot \frac{EB_{RAM}}{1000000}\right) \pm \left(0 \cdot \frac{EB_{RAM}}{1000000}\right)\right) FR$$

$$FIT_{RAM}\left(\frac{FIT}{Mb}\right) := (\pm 0)\ FR \tag{2.3.7}$$

$$FIT_{FF}\left(\frac{FIT}{Mb}\right) := \left(\left(1.96 \cdot \frac{EB_{FF}}{1000000}\right) \pm \left(0.54 \cdot \frac{EB_{FF}}{1000000}\right)\right) FR$$

$$FIT_{FF}\left(\frac{FIT}{Mb}\right) := (0.001591520000 \pm 0.0004384800000)\ FR \tag{2.3.8}$$

$$FIT_{DUT}\left(\frac{FIT}{Mb}\right) := \left(\left(9.447608059 \cdot \frac{EB_{DUT}}{1000000}\right) \pm \left(0.1505932201 \cdot \frac{EB_{DUT}}{1000000}\right)\right) \cdot FR$$

$$FIT_{DUT}\left(\frac{FIT}{Mb}\right) := (8.406207670 \pm 0.1339934800)\ FR \tag{2.3.9}$$

Figure A.21: Calculations for the Design with TMRs and EDAC plus Device TMR, 4 faults.

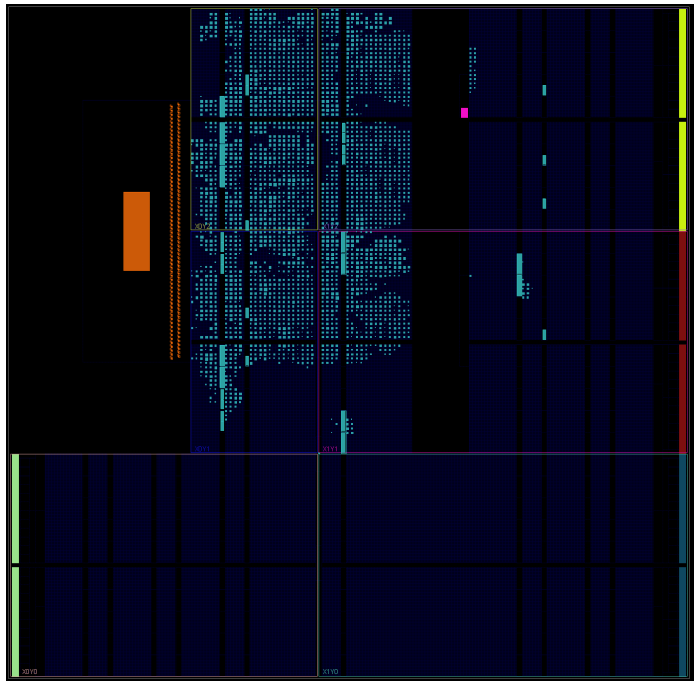

Figure A.22: Floor-Plan for the Design with TMRs and EDAC plus Device TMR.

Resource	Utilization	Available	Utilization %
LUT	10305	53200	19.37
LUTRAM	1	17400	0.01
FF	2361	106400	2.22
BRAM	18	140	12.86
DSP	9	220	4.09
BUFG	1	32	3.13

Figure A.23: Resources employed by the Design with TMRs and EDAC plus Device TMR.

Bibliography

[1] K. Abdul-Rafay, N. Nasreen, and A. Ehsan. "Redundant Techniques for FPGA-based Designs: A Technical Survey". In: *International Journal of Computer Science and Information Security (IJCSIS)* 15.6 (June 2017). ISSN: 1947-5500.

[2] A. Avizienis et al. "Basic concepts and taxonomy of dependable and secure computing". In: *IEEE Transactions on Dependable and Secure Computing* 1.1 (January 2004), pages 11–33. ISSN: 1545-5971. DOI: 10.1109/TDSC.2004.2.

[3] M. Bagatin, S. Gerardin, and A. Paccagnella. "Ionizing radiation effects in electronic devices with an emphasis on non-volatile memories". In: *2015 IEEE International Integrated Reliability Workshop (IIRW)* (2015).

[4] A. Gerald Stoddard. *Configuration Scrubbing Architectures for High-Reliability FPGA Systems*.

[5] R. Glein and F. Rittner. *Reliability of Space-Grade vs. COTS SRAM-Based FPGA in N-Modular Redundancy*.

[6] P. Graham, M. Caffrey, and J. Zimmerman. *Consequences and Categories of SRAM FPGA Configuration SEUs*.

[7] Paul Graham et al. "Consequences and categories of SRAM FPGA configuration SEUs". In: *Proc. 5th Annu. Int. Conf. Military Aerosp. Program. Logic Devices* (January 2003).

[8] I. Herrera-Alzu and M. Lopez-Vallejo. "Design Techniques for Xilinx Virtex FPGA Configuration Memory Scrubbers". In: *IEEE Transactions on Nuclear Science* 60.1 (February 2013), pages 376–385. ISSN: 0018-9499.

[9] J.W. Howard and D.M. Hardage. *Spacecraft Environments Interactions: Space Radiation and Its Effects on Electronic Systems*.

[10] F. Lima Kastensmidt, L. Carro, and R. Reis. *Fault-Tolerance Techniques for SRAM-based FPGAs*. 1st edition. Springer, 2006.

[11] L. Philip H.W. *Recent Trends in FPGA Architectures and Applications*.

[12] A. Ruan et al. "Insight Into a Generic Interconnect Resource Model for Xilinx Virtex and Spartan Series FPGAs". In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 60.11 (November 2013), pages 801–805. ISSN: 1549-7747.

[13] Usman Sani and Ibrahim Shanono. *Design of (7, 4) Hamming Encoder and Decoder Using VHDL*. 2015.

[14] Dr. Juergen Sauermann. *How to design your own CPU on FPGAs with VHDL*. 2010.

[15] E.G. Stassinopoulos and K.A. Label. *The Near-Earth Space Radiation Environment for Electronics*.

[16] C. Stroud et al. "BIST-based diagnosis of FPGA interconnect". In: *Proceedings. International Test Conference* (October 2002), pages 618–627. ISSN: 1089-3539. DOI: 10.1109/TEST.2002.1041813.

[17] I. Tuzov, D. de Andrés, and J.C. Ruiz. *Accurate robustness assessment of HDL models through iterative statistical fault injection*.

[18] I. Tuzov, D. de Andrés, and J. Ruiz. "DAVOS: EDA Toolkit for Dependability Assessment, Verification, Optimisation and Selection of Hardware Models". In: *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. June 2018, pages 322–329. DOI: 10.1109/DSN.2018.00042.

[19] M. Wirthlin. *High-Reliability FPGA-Based Systems: Space, High-Energy Physics, and Beyond*.

[20] Xilinx. *7 Series FPGAs CLB User Guide*. 2016.

[21] Xilinx. *Device Reliability Repor*. 2019.

[22] Xilinx. *Series FPGAs Configuration User Guide*. 2015.