Observability of off-the-shelf microarchitectures based on the RISC-V Instruction Set Architecture

Wouter van Huffelen, *student, EMSYS University of Twente*

***Abstract*—Fast and low-cost development is becoming more important for the space industry with a greater need for communication infrastructure in space. Commercially off-the-shelf (COTS) components already have been widely used in industry, and with the raise of RISC-V core architectures as off-the-self core that can be deployed. How can we evaluate if a core is reliable in hazardous conditions? To characterize these cores, this paper looks at the observability problem within irradiation campaigns. With the use of fault injection in simulation and error modeling, an approximation is made to characterize the Ibex RISC-V core and its behavior before the real experiment. The experience and data from the irradiation campaign are then used to improve and verify the simulation strategy. The preparation for the experiment is faced with many practical challenges that are less dominant in simulation. The biggest bottleneck is the interface and transmission of data between the Device under test (DUT) and the host observer.**

***Index Terms*—RISC-V, Irradiation, FPGA.**

I. Introduction

THE space race is back on the agenda, especially now that commercial companies are competing too. In time, this will increase the demand for electronics suitable for space. Space-grade components have long development cycles and limited production numbers, which result in high costs per unit. To mediate the high cost of these components, companies will opt for the use of Commercially off-the-shelf (COTS) components. Using a general purpose Field Programable Gate Array (FPGA) and deploying an optimized microarchitecture, it is possible to reduce the cost. Especially now with the rise of RISC-V Instruction Set Architecture (ISA).

Space is a hazardous environment with high levels of radiation, roughly 25 times the radiation level in low earth orbit than on earth[17]. The impact of ionizing particles can corrupt signals and data in the form of Single Event Effects (SEE). A famous example of a particle interacting with our world is the Super Mario 64 glitch [6]. The last thing companies want is for their satellite to crash because of SEE corrupting data, and this makes reliability very important. Once the hardware is up at "space", it is very difficult, near impossible, to physically access satellites. To ensure correct operation in a hazardous situation, proper testing must be performed through irradiation campaigns and simulations. There are several examples [9][3][15] of processors that implement multiple reliability techniques (e.g., Triple Modular Redundancy, Error Correcting Code or Physical Memory Protection) with the goal of minimizing single points of failure in the system. However, the effectiveness shows dubious results on the source of the improvement. The easy way out is to use every redundancy scheme to make the system as reliable as possible. Area and energy optimizations are disregarded by applying redundancy schemes without a clear motivation of the effectiveness. RISC-V ISA allows for control and modularity in contrast to proprietary ISAs. This results in companies and research groups developing their own RISC-V microarchitectures, which can become COTS soft-core architectures for other users. The sheer benefit of RISC-V is twofold: RISC-V is open and modular. It has a community backing it, so it is alive and thriving contrary to something like SPARK. Depending on the developer and licensing, the developed core can be modified and extended where needed. These practices can significantly reduce development times in the downstream.

The current methods for improving redundancy and reliability can be improved upon. This requires a deeper understanding of the impact of particles on systems. Most existing studies present only success stories, lacking a comprehensive evaluation of their methodology. To address this issue, the observability of Single Event Effects (SEE) is being increased to reduce the survival bias. The question is whether highly efficient fault-tolerant schemes can be introduced in a RISC-V Core through improved observability of Single Event Effects (SEE). To answer this question, the use of fault injection in simulation is proposed using a research core built using a RISC-V Instruction Set Architecture (ISA), which allows for hardware access. The goal is to study the root causes of issues introduced by SEEs, not to build a commercial RISC-V core. To validate different redundancy and security schemes, a development framework will be established that can mimic the complexity of the irradiation experiment. The simulation, through the use of pseudorandom fault injection, will increase observability during the development phase and aid in understanding the faults, making the development of a fault-tolerant scheme for a RISC-V core more efficient over time. The comparison between the simulated and real experiments will facilitate the development of better fault-tolerant schemes.

This report presents a comprehensive study on the application of pseudorandom fault injection in the development of a fault-tolerant RISC-V core. The report is structured as follows: Chapter II provides the background information related to the field, including the ionizing effects on silicon, the problems that can be caused by irradiation, an overview of the RISC-V architectures, and a discussion of common redundancy schemes. Chapter III focuses on the related work that touches on the above questions. Chapter IV introduces fault injection and simulation as an important tool in the development of reliable processors. This chapter includes a discussion of the fault modelling used for the experiment and

the observations from the simulation. Chapter V presents the error modelling process and the methodology used to model errors in the system. Chapter VI presents the setup for the beam experiments and the results. Chapter VII provides the conclusions, summarizing the key observations made during the study and making recommendations for future research endeavours in this area.

II. Background

A. Where did it begin?

It is a big cliché by now, but it still holds. It all started with a big bang many, many years ago. There are multiple sources of radiation in the universe, the star called the sun is the nearest emitter of radiation. Cosmic rays that are created during events in space, like from a supernova, also eventually reach the earth and scatter on impact with the electromagnetic field of the earth as is shown in Figure 1. The resulting ionized particles collide and impact with other particles; this phenomenon is generally known as Background Radiation (BR). The BR increases with altitude, as the shielding of the gets weaker away from the core [17]. In small doses this BR is harmless, but with greater exposure, the resulting effects can be deadly.

A famous example of a particle interacting with circuitry is the Super Mario 64 glitch [6]. During a speed run of Super Mario 64, an unexpected movement was made by the character that resulted in skipping a section of the game. The investigators suspect that it was Silent Data Coruption (SDC) in the height positioning register due to BR, all attempts to reproduce the bit overflow were in unsuccessful. This example [6] shows what ionized particles could do to electrical hardware and is therefore an iconic example of a Single Event Upset (SEU).

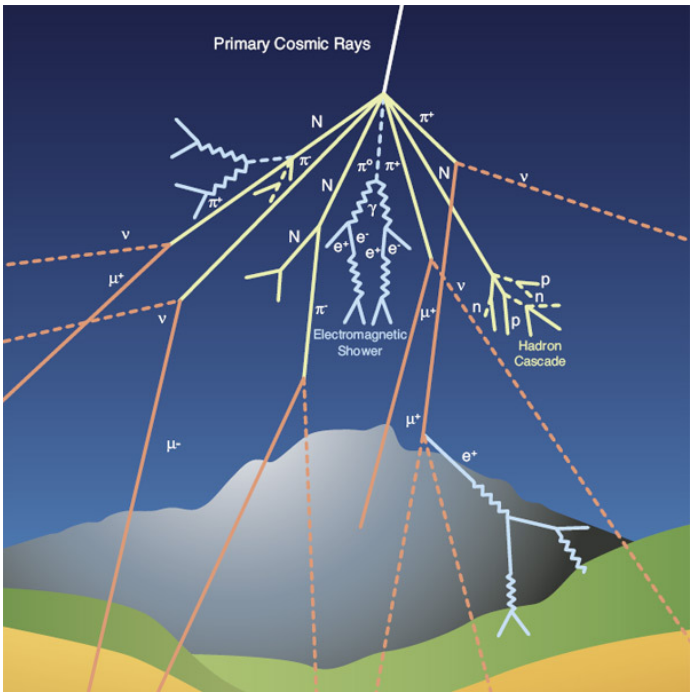


Fig. 1: Source CERN [7], a cosmic ray entering earth's atmosphere and scattering in different particles.

If our electronics are susceptible to background radiation, imagine what would happen to a device that is in direct exposure to cosmic rays. Or what would happen to mission-critical infrastructure if SEU disrupts calculation and communications? The current world highly depends on functioning information infrastructure (server farms, satellites, antennas, etc.). But concerns about SEUs are not limited to communication technology. A more direct example would be "self-driving automobile". The societal impact of a vehicle hitting a pedestrian due to SDC would be immense. Therefore, it is important to implement irradiation campaigns on hardware to test for faults. Irradiation campaigns are not new; years of research have been done in this field. Most of the irradiation campaigns are aimed at material science and how materials react. This research has led to specially treated rad-hardened silicon that mediates the effects of these particles. Manufacturers of microcontrollers and Field Programable Gate Array (FPGA) often support their space-graded products with features like Triple Modular Redundancy, Error Correcting Code memory, routing optimization, etc. The White Paper [37] gives an overview of a product intended for space applications. The paper shows that the company is prepared to invest resources to create a COTS device that can be used by space agencies, such as European Space Agency (ESA), for their satellites and rockets. The takeaway here is that with only rad hardening of the silicon, reliability is not ensured and, in any case, it would not be economically viable.

Now that it is established that radiation is bad for chips as well as for humans, a closer look at the reasons why it goes wrong. The interaction between the particle and devices is studied with material science. The short explanation is that the base material from which the transistor gates are created is affected by the ionized particle. Logic gates are built out of differently doped silicon to create junctions based on energy potential. The ionized particle can cause a state change or, occasionally, alter the properties of the material temporarily or permanently. Generally, these are referred to as Single Event Effects. If the state is changed temporarily, then it is called a Single Event Transient (SET). The SET can have lasting effects in the register of memory, which are called Single Event Upset (SEU). NASA did a lot of research on this topic [26].

It is important to note that over the years the gates and junctions have gotten smaller, increasing their amount on a chip. This has led to a higher susceptibility to particles. The density and size of these gates in Integrated Circuit (IC) is a disadvantage to the vulnerability of the chip. That is why one of the common techniques to make chips more resilient is to use a bigger production process of 26nm[1] to 22nm instead of the 7nm to 5nm process that is used to manufacture state-of-the-art chips.

SEE are not always an issue. A SET goes unnoticed if parts of the circuit are clock-gated or not used at the time of impact. Most logic circuits are timed and therefore less sensitive to disturbance in the circuit. Circuits that are used to store states, like flip-flops and registers, are very vulnerable.

[1]Nanometer (nm) is $1 * 10^{-9}$ meter

Studies show that memory is the greatest source of SDC [28] [20]. The severity of the effects of SEUs depends on the type of memory. For example, DRAM memory uses a charge of a capacitor to store data. The mechanism can trigger and lose its original charge and bit-flip if the impact of the radiation is at the mechanism that keeps the charge in the capacitor.

Through the use of ECC most of the SEU in the memory can be corrected. But as [27] shows, ECC alone does not cover Mutiple Bit Upsets (MBU). [27] points out that the number of critical errors relatively increases when using memory protection. These errors are assumed to be happening in the core architecture itself. As the core stops functioning, the program crashes without warning or detection. The common way to protect the core of SDC is to build redundancy in the shape of double or triple redundancies of the core. The most used method for processor units is to apply Triple Modular Redundancy (TMR) with a voter. Computing the result three times and choosing the equal results, leads to the correct result. However, this increases the cost of the hardware in monetary value, space, and energy use. The papers [2], [31], [36] and [35] all suggest a form of TMR to protect the device from SDC or fatal errors. Most of the solutions in these papers combine TMR with ECC, as this combination already gives a significant performance boost in reliability. What really happened inside the core remains unanswered. Most of the papers only cover output comparisons and assume an understanding of the probable fault triggered. The authors of [29] give a good overview of different solutions for different kinds of issues, based on where the SEU is anticipated to happen.

In the above-mentioned papers, they use metrics such as fluence and cross-section, to denote the power of the irradiation beam and the characteristics respectively. It is important to remember that there is a chance that a particle in a given area interacts with the device. The energy of the irradiated particles in an area is called flux. The exposure of flux (Φ_e, eq.1) over a given time is called fluence or radiant exposure (H_e, eq.3). The cross-section is the number of errors in the device divided by the fluence (cs, eq.4).

$$\Phi_e = \frac{J}{s} \qquad [W] \tag{1}$$

$$E_e = \frac{\Phi_e}{A} \qquad [\frac{W}{m^2}] \tag{2}$$

$$H_e = \int_0^T E_e(t)\,dt \qquad [J/m^2] \tag{3}$$

$$\text{cs} = \frac{\sum error}{H_e} \tag{4}$$

The cross-section of the device is meant to illustrate its sensitivity to radiation dose, as determined by the total number of errors detected within a specific measurement timeframe. However, this raises the question of the undetected errors and their impact on the efficiency of fault-tolerant measures. To address this, it may be worth considering the approach taken by Abraham Wald [14], who used the concept of survivorship bias to understand why some warplanes did not return from battle. He observed that the surviving aircraft were not damaged in critical systems, allowing them to return home safely. This led him to conclude that the parts that had not sustained damage should be reinforced. This analogy is particularly relevant to embedded systems operating in space, where not all damage (SEE) may be critical or observable. In the event of a non-responding chip, very little information can be salvaged. While much of the fault and error detection and correction occurs in software, critical failures cannot be recovered if the system fails to respond.

B. RISC-V

The RISC-V Instruction Set Architecture is introduced in 2010, from that point many papers have been written on it. There is no point in fully describing how this ISA came to life or how it works, as a dedicated paper would do it more justice. What will be touched on is what is RISC-V, why is it interesting and how is it already deployed? For the person who likes to read up on the history of RISC-V, please visit the webpage of the foundation itself [30].

RISC-V is, as its name implies, a Reduced instruction set computer (RISC) ISA, and therefore has fewer instructions than a Complex instruction set computer (CISC). The most well known CISC is the X86 ISA, as for the RISC is the ARM platform. The pros and cons to choose a RISC processor over a CISC depends on its applications. In [12] compelling arguments are given for the application of RISC-V in space. The key argument in the article for RISC-V is open-source. The research institution released the RISC-V ISA under the Berkeley Software Distribution (BSD) license. In contrast to other architectures, like mentioned before, the RISC-V is open and can be used - under conditions of the BSD license - without “black-box” Interlactual Property (IP). The best example of a closed off IP is ARM core, as for the integration of the core into a product licensing fees will have to be paid. If a customer desires a custom ARM core, additional costs will be added to the licensing subscription. But an open-source ISA does not mean that every RISC-V core design is public for use. Companies are still allowed to protect and sell their version of a RISC-V architecture.

For this research a RISC-V research core has been chosen. The chosen core is the Ibex core from the PULP platform [11]. The reasons for choosing this architecture are based on complexity, support, openness/accessibility, and area of use. It is a competent core that can be easily pictured in an embedded system for space. There were other contenders, for example, the bigger brother of the Ibex the CV32E40P [15], or a core of the Klessydra Core Family. There are also cores of companies available, but they were not considered as they are not fully open and as being restricted to a partnership doesn’t fit the goal of this research. The Ibex is a simple two-stage core that reduces the number of intermediate registers between stages, see Figure 2. The benefit of a smaller core is that alterations in the core are easier made if needed. Because of the two-stage design, faults in the core show up quicker at the output. The other reason is the level of support, the PULP platform is well-maintained and reasonably well-documented. The PULP project, like the Ibex, already has turn-key versions in them for use.

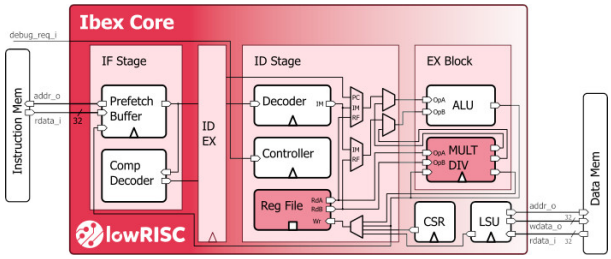


Fig. 2: Block diagram view of the Lowrisc Ibex core, provided by Lowrisc

C. Redundancy & Security

Throughout this paper, numerous terms and concepts have been introduced. However, it has become evident from the literature that three subjects require further elaboration as they are critical to the paper's understanding. These include Memory, followed by TMR, and finally, addressing certain security concerns.

1) Memory: Memory is a critical component in modern computing that provides a way of storing and retrieving data and instructions for processors to use. Different types of memory have different fabrication processes and can affect their susceptibility to SEE.

There are two broad categories of memory: volatile and non-volatile. Volatile memory requires a constant power supply to retain data and will lose all stored information if the power is turned off. Dynamic Random Access Memory (DRAM) and Static Random Access Memory (SRAM) are both examples of volatile memory. Dynamic Random Access Memory (DRAM) is widely used but is particularly susceptible to Single Event Effects (SEE) because it needs constant refreshing to maintain the charge in its capacitors. On the other hand, Static Random Access Memory (SRAM) stores data as the state of a flip-flop circuit and does not require constant refreshing, making it faster than Dynamic Random Access Memory (DRAM) but also more expensive.

Non-volatile memory, on the other hand, retains its data even when the power is turned off. Flash memory, also known as NAND flash, and Read-Only Memory (ROM) are both examples of non-volatile memory. Flash memory is widely used in portable devices as an economical alternative to Static Random Access Memory (SRAM) and Dynamic Random Access Memory (DRAM).

To improve the reliability of memory, techniques such as ECC and Physical Memory Protection (PMP) are widely used. Error Correcting Code (ECC) uses a form of Hamming code to detect and correct up to one-bit errors and detect two-bit errors, providing a layer of protection against Single Event Effects and reducing the risk of data corruption. PMP, part of the RISC-V Privileged Architecture Specification, offers hardware protection of memory regions that processors cannot access and throws an exception when unauthorized memory space is requested by the core, further reducing the risk of data corruption.

2) Triple Modular Redundancy: Triple Modular Redundancy, or TMR, is a technique used in digital electronics to increase the reliability and fault tolerance of a system. TMR involves the use of multiple identical modules to perform the same task, with the outputs of the modules being compared and the majority output being selected as the correct answer.

Before diving into TMR, it's worth mentioning Dual Modular Redundancy (DMR), which is a similar technique that involves using two identical modules to perform the same task. A good analogy of DMR is the use of a backup generator in a hospital. While DMR can provide some level of fault tolerance, it is not as reliable as TMR. In the case of a hospital, the backup generator is only used as a pure backup in the event that the main generator fails. This is an example of two times redundancy.

TMR provides an extra module and a voter, which is crucial in ensuring that any error or fault in one of the modules is detected and corrected by the majority of the modules. The voter operates based on the principle of majority voting, with the outputs of the three modules being compared and the majority output being selected as the correct answer.

However, TMR comes with some associated costs, mainly in terms of chip area and power consumption. There are reports that claim to have achieved minimal cost increases for TMR, but the size usually increases by a factor of three. Despite this, there are innovative solutions that attempt to optimize the size increase by enabling TMR to serve as parallel computing cores, maximizing both reliability and performance.

It should be noted that TMR is not a universal solution. If the input data is already corrupted by SEUs, then it is certain that the output will also be corrupted, as TMR only provides fault tolerance and error detection and correction, not error prevention.

3) Backdoors: For observation purposes, direct access to the core can provide valuable insights, but it also poses a significant security risk. The existence of a backdoor, which allows unauthorized access, is a major concern in the current cybersecurity landscape. This concern was heightened following the discovery of the Meltdown and Spectre security vulnerabilities in 2018 [19, 22]. These vulnerabilities, which exploit weaknesses in the design of modern computer processors, underscored the importance of considering security in the design of hardware solutions.This is because the memory manipulation techniques used by Spectre and Meltdown, which were implemented by the designers of the chips to speed up processing, are the same techniques that can be exploited for malicious purposes.

Injecting instructions into the core, or the ability to read them, could have serious consequences and must be avoided to protect the integrity of the system. Any hardware solution must be designed with security in mind, taking into account the potential risks and the need to maintain the confidentiality and privacy of sensitive information while balancing this with the necessary level of observability.

III. Related Works

In previous chapters, many papers relevant to the research topic have been discussed. Therefore, there is no need to repeat them in this chapter. Instead, a few new papers that have not

been previously mentioned will be addressed. These papers are significant as they shed light on the research question in new and unique ways.

A. DIVA & Argus

Already in 1999, researchers were working on checking errors online in executing code, Dynamic Implementation Verification Architecture (DIVA) [4]. DIVA was a proposal for a bridge between hardware and software based on the TMR idea. By adding additional hardware to the core and utilizing special encoding, DIVA can detect errors in the control-flow or execution of the core. In 2007 an improvement proposal has been made with Argus [24]. Argus main improvement is the addition of checks on Memory and Dataflow. The creators of Argus showed that it also can be used to pinpoint where the errors happen [25]. Recently, this technology popped up again in papers like [10] and [1] where they look at the observation question for their designs. With RISC-V dynamic verification methods may be smart features to add. The actual benefits still have to be tested, but it may be a promising endeavour.

B. FIRECAP

A great idea that encapsulates the spirit of adding hardware to increase observability is FIRECAP, co-developed by STMicroelectronics [33]. FIRECAP is an add-on to the core that hooks into vital registers and keeps track of the progression of the core. When the core stops responding, the programmer can access FIRECAP through the debug interface to retrieve valuable information on what happened to the core. Additionally, FIRECAP has the ability to inject instructions into the core from its own memory, making it a more powerful tool for observability. However, FIRECAP is subject to radiation and may not always be reliable. The researchers have addressed this issue by adding proven mediation technologies, such as TMR and the use of different-sized production methods, to improve its reliability. In essence, they have created a hardware solution to the observability problem.

C. SEM IP

The Soft Error Mitigation (SEM) IP is a module developed by Xilinx to combat the main weakness of their SRAM FPGAs. The main problem in SRAM is that the design of the hardware is in memory and memory is the most vulnerable to irradiation. Having your hardware design in memory causes that the data as well as the design of the architecture is subject to change. Fortunately, the configuration is a data file and therefore error checking methods can be used to determine and even fix bits in the configuration. Xilinx does not claim that the IP can resolve MBU or all the SBUs. The SEM IP has three modes of operation: detect upset, fix upset and fix or replace upset. The fix or replace upset mode is meant to be the solution to non-recoverable errors as it re-configures the FPGA from protected memory [5, 23]. In Chapter VI the SEM IP will be used as it is critical for the experiment.

IV. Fault Injection

To prepare for irradiation experiments, simulations have been conducted to observe the behaviour of the system's micro-architectures. These simulations aim to identify weaknesses in the architecture and characterize resulting errors and mistakes through fault injection. Most simulations use test cases designed to check functionality, which limits their scope to a subset of real-world equivalent. That is why a different approach had to be taken to the simulation in order to test different redundancy schemes and their effectiveness.

After the real experiment, the simulation will be done again. The idea for redoing the simulation is to update the understanding of the current simulation and consider its value. Furthermore, it is plausible that parameters, designs, or code have changed during the experiment that needs to be re-evaluated. The goal of the post-simulation phase is to attempt to reconstruct the found errors.

A. Simulation environment

1) Tools: The tools were mostly dictated by the chosen soft-core architecture. The maintainers of the Ibex core used FuseSoc [18] to manage their project and Verilator [32] as their main way of simulation. One of the core reasons that Verilator is used is that the Ibex is programmed in SystemVerilog. Verilator is the fastest Verilog/SystemVerlog simulator compared to Modelsim™.

2) Fault-injection mechanism: Verilator allows the use of a modified C++ script to perform fault injection. During the fault injection, logic/wires are targeted. In order to target the logic, knowledge is needed of the simulated design. The biggest drawback of this method is that the entire design has to be recompiled if a different point in the system wants to be targeted than the compiled target points. It is possible to target multiple points per simulation. As the goal is to simulate SEEs, the script is instructed to change only one bit at a pseudorandom point in time of the simulation.

3) Evaluated Benchmark: The goal of this simulation, and later the experiment, is to identify the origin of each observed SDC. Therefore, it has to be possible to retrieve the computed result and backtrack it to the origin of the errors. In standard benchmarks, this information is usually either disregarded or is difficult to extract, as the focus is to retrieve the result of a compute-heavy workload on the core. For a microcontroller, a more realistic scenario is one where data is computed as it arrives, and the processor operates around the necessary IO of the system. So, instead of investigating multiple benchmarks, the choice to evaluate a mixed application that *computes and transmits* data was made. Thus, the benchmark is composed of a workload that modifies data, followed by a period where the processor is waiting for external events (e.g., waiting for an interrupt), which in this case it is represented by the communication via the UART.

Based on the investigations of previous works [34, 21], a 15x15 matrix multiplication (32-bit word) is selected as the workload. This choice is advantageous for two main reasons. Firstly, it is a common workload evaluated fairly often on similar works [13, 21], with a decent memory and computing

footprint. Secondly, it allows for backtracing changes in the output, so where an error occurred in the kernel (input data, computation, or output data) can pinpoint. This will be further explained in SectionV.

Listing 1: Computing Kernel

```
uint32_t matrix_multiply(void){

    uint32_t i,j,k;

    for(i=0;i<ROWS;i++)
    {
        for(j=0;j<COLS;j++)
        {
            c[i][j]=0;
            for(k=0;k<COLS;k++)
            {
                c[i][j]+=a[i][k]*b[k][j];
                c[i][j]+=a[i][k]/5;
                c[i][j]+=b[i][k]/5;
            }
        }
    }

    return 0;

}
```

B. Results

The decision to use the Lowrisc Ibex core was explained in Chapter II-B, and one of its advantages is the RISC-V Formal Interface (RVFI) backdoor, which enables formal verification. During simulation, the tracer module can utilize the RVFI to display all executed instructions and states by the Ibex core. However, the tracer module is non-synthesizable, which makes it unsuitable for actual synthesis. Through internal observation via the RVFI, it is possible to trace the corruption of control logic or data. For instance, an error in the initial pointer to a register can lead to an ongoing jump offset in the program, which is a type of control logic corruption that can be categorized as a register file error. Another example is the branch prediction and branch taken registers, which can produce delayed errors over 2 to 6 clock cycles if their states change well before they are needed.

For the Ibex core, over 200 logic points were tested. The majority, approximately 88%, of the tested points did not result in any visible changes in the output or execution trace. This could indicate that they were part of the executed operation, and therefore deemed clean and/or non-critical. The remaining 12% can be divided into three groups: detected (3%), not detected (3%), and aborted by simulation (6%). The predominant source of critical errors are found in the ID- and IF-stage of the device. These errors are mostly Register file errors, observed like the example in the paraghaph above. Exceptions thrown by the core are suspected to be caused by incorrect program counter trying to access invalid memory spaces. Sometimes the core itself report this violation, as long the exaction handler is correctly programmed. This observation results in a methodology presented in the next chapter.

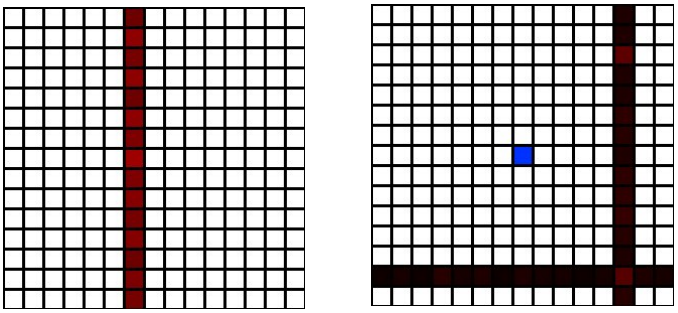
Simulation is a valuable tool for creating, preparing, and testing ideas. In today's business world, companies often use simulation tools to optimize costs and improve the efficiency of their prototyping processes. The goal is to get it right the first time, and simulation can help achieve that. However, it's important to note that simulations can never perfectly replicate the real world. The accuracy of a simulation depends on the quality of the tools used and the test cases designed by the engineer. Most test cases are designed to check functionality, which means that the simulation may only represent a subset of the real-world equivalent. Despite these limitations, simulations offer several benefits. For one, they provide complete observability within the constraints of the tools used. Additionally, they can help designers rapidly prototype and reduce costs.

V. Error modelleing

To correctly identify the origin of the observed errors, it is necessary to pinpoint the root cause of the abnormal behavior, seen in the data. With this goal, a model to differentiate data corruption errors based on their effect on the system, Hamming Distance (HD), and longevity (if the error carries to the next loop iteration) is used:

User input data: In accordance with Section IV-A3, the chosen kernel in this process is specifically selected due to its ability to precisely identify error locations during computation. The change can graphically be visualized, as shown in Figure 3. An error in a word of the input data will propagate to the rest of the matrix as either an error in a column (Figure 3a) or a column and a row (Figure 3b). The different colors in each cell represent the HD between the output and the golden value (blacker is more distant, redder is closer). A single bit-flip in the input data will cascade a significant change for each cell. If the bit-flip happens in the output data, the error will not cascade and remain unique. Thus, it is possible to differentiate errors that only occur in the output matrix (blue square in Figure 3b). Errors that happen in the register file during computation will affect a single cell with a HD > 1 and errors in one of the constants will affect the entire matrix. Furthermore, input data that is modified (i.e., A and B matrices) will not be overwritten until the processor is reprogrammed. It is expected to see a build-up of errors that remain stable through iterations.

Communication error: Because the benchmark also has a communication mechanism, it is needed to account for SEUs that might also occur there. Possible sources of issues are control flow errors in the parsing of the output matrix and in the encoding of the ASCII characters. The output rests in memory until the next computation, so errors can still build up in the output until they are transmitted. Additionally, an encoded ASCII character will remain in memory until there is space in the buffer to send it, thus it can also accumulate errors. A simple way to separate these issues is to filter results that don't result in an ASCII hexadecimal numeric value (e.g., the character *"?"* has a HD of 1 compared to *"7"*). It is more



(a) Error in input a (b) Error in b and in c

Fig. 3: Propagation of errors in the output matrix.

difficult to classify errors that provide sane values with a HD of 1, they are classify as errors in the output matrix.

Register file errors: This is the most volatile data, with only the Stack Pointer (sp), Global Pointer (gp), return address (ra), and a saved register (s0) remaining alive during the entire execution of the loop (albeit sometimes moving to the stack). A SEU in any of these would result in invalid memory access, causing a Detected Unrecoverable Error (DUE). The only other registers used in the application are registers a0-a5 which are rewritten constantly. These could cause SEUs to propagate and create changes with a HD > 1 or control flow issues.

Programming logic: Programming logic bits are only ever rewritten when the device is reprogrammed. Thus the effects will accumulate. Although not all SEUs in this category will be detectable, some of them will result in DUEs or SDCs. Specifically for SDCs, it is expected that they will create new stable output states.

Critically, the model is trying to filter multi-bit and multi-position errors to a single bit-flip. These SEUs might even remain persistent between iterations, even if they do not originate just from Configuration RAM (CRAM). As explained above, SEUs in the data memory on the input data will be seen as permanent errors, even though they were caused by a single bit-flip. To counter that, the model counts only new errors. When an input data fault occurs, it will create a new stable output state, these variations are captured and the golden matrix is update to match the new state. This strategy also works for errors in the configuration memory that can produce a stable output (further described in Section VI-C1). To trace communication errors, the same strategy for errors in multiple, but constant, cells (i.e., a permanent change to a character) can be used, for transient errors different criteria will be needed. As described above, SEUs in the communication step can have a non-sane character value or a HD of 1 in an individual cell. Together with the new error strategy, these 3 criteria compose a parser that filters all of these errors in a post-processing step. Exceptions to these criteria (described in Section VI-C1) are marked for manual inspection.

VI. Irradiation Campaign

A real-life experiment was conducted at the ChipIr facility at the STFC Rutherford Appleton Laboratory in Oxfordshire to gain experience with the irradiation campaign. Despite unforeseen events hindering the realization of the original plan, the experience gained was crucial in understanding the radiation campaign and unexpected results were obtained. The plan was altered several times due to practical constraints. Initially, a Flash-based FPGA was intended to be used, but due to a global chip shortage caused by the COVID-19 pandemic and limitations of the development tools provided by the flash-based FPGA manufacturer, the decision was made to switch to a SRAM-based FPGA from Xilinx, which offered a feature that would aid in the experiment and had the most modern and reliable development tool on the market.

A. Physical setup

The selected chip from Xilinx was the XC7A100TCSG324-1. The chip is available on the development board ArtyA7. The ArtyA7 has a reasonable community around it that gives support in forms online. It is sometimes overlooked, but documentation is the key and more information is available, the easier it is to adopt a certain product. One oversight was that the ArtyA7 board does not have proper mounting holes, what made it less ideal for mounting with tie wraps.

Figure 4 depicts the experimental setup. The DUT was placed in the beam line and connected through high-speed USB replicators to the host PC in the control room. The processor computed and transmitted the data as soon as it was ready. The host PC was responsible for reprogramming the device and storing the data safely. After the experiment, the data was processed offline.

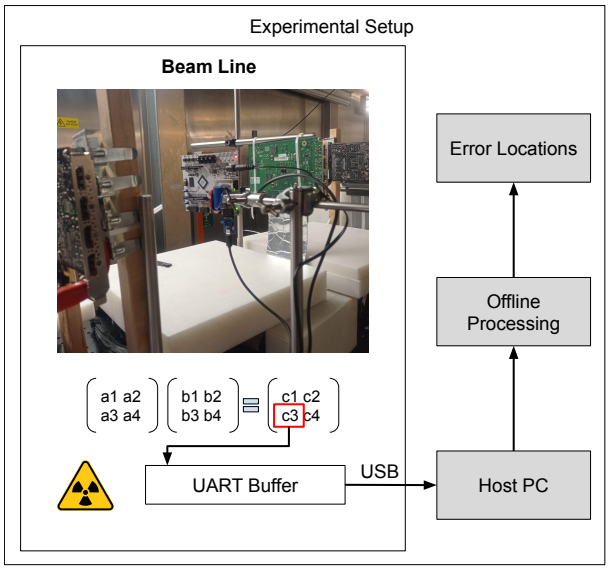


Fig. 4: Experimental setup

This specific implementation was adapted from the Ibex Super System project[8], which couples the Ibex with a debug module, an UART, a GPIO, and a timer in a System-on-Chip (SoC). The design's post-place-and-route resource utilization used a total of 4564 logic LUTs, 96 distributed RAM LUTs, 3138 FFs, and 32 blocks of RAMB18 block ram. All the available BRAMs on the target device were utilized. The register file and inter-stage buffers were mapped

to distributed RAM (LUTs). The benchmark itself was on an infinite loop, but regardless of the current status, the host PC was tasked with relaunching the application every 10 minutes to avoid accumulating too many errors on a broken output. This procedure reprograms both the user and configuration memories.

In Figure 5, the fluence of the beam during the long-running experiment is shown. It can be seen that there are two time periods that the fluence went to zero. The exact reason why the fluence went to zero is unknown, but it is not an uncommon phenomenon as it is difficult to produce a constant fluance. The beam at the STFC Rutherford Appleton Laboratory is well known for a mostly consistent beam. The measurements during those period of low/zero fluency, could be discarded, but are used as control group measurements of the DUT. The chip itself becomes radioactive overtime as it gets irradiated. Therefore, it is good to have in between measurements without external radiation.

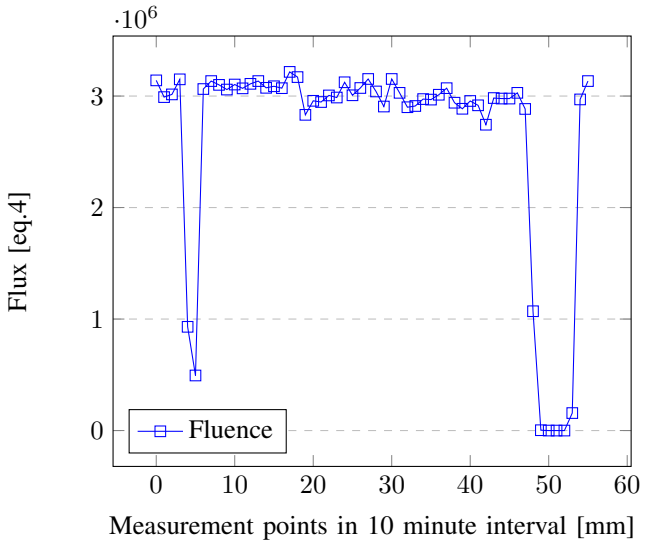


Fig. 5: Measured flux of the beam line during the main experiment, starting at 11:45:21 PM on 26th of Sep.

B. SEM IP

The original plan had to be carried out in two steps. The first step was to check the working of the SEM IP on its own. As the choice was made to switch to a SRAM based FPGA, because of practical reasons, the configuration memory had to be taken into account. As was mentioned before, SRAM based FPGAs suffer more SDC compared to flash based. In order to know which SDC comes from the program itself and which follows from the configuration, the cross section of the SRAM FPGA had to be characterized. The plan was first to expose only the SEM IP to the beam to get this result. Under close observation the test was run multiple times, whereby the configuration was uploaded and observed till the SEM IP reported a non-recoverable state. When the non-recoverable state was observed, the configuration was scrubbed from the device and compared to the original bitstream.

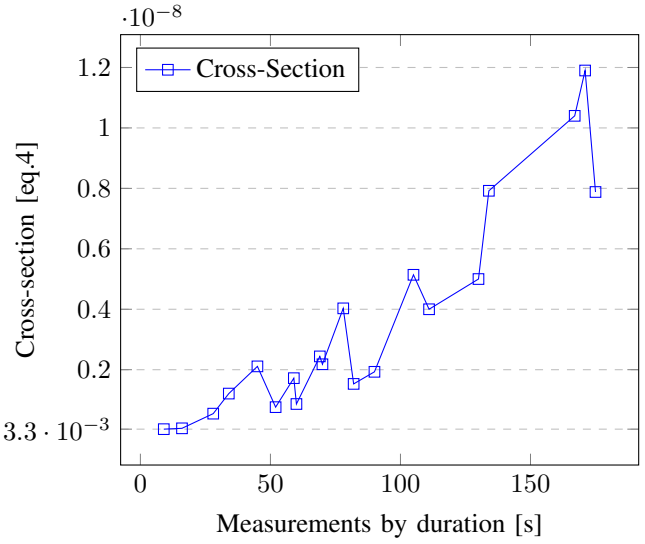


Fig. 6: Cross-section on SEM IP reported errors

In the Figure 6, the cross-section deducted by the reported error of the SEM IP is given. The cross-section increases the longer the SEM core is operational.

The second step is to test the microarchitecture of the Ibex with the SEM IP. The original idea was to test one device with SEM IP and one without. Unfortunately, the SEM IP did not integrate with the microarchitecture correctly, which may be due to a fault that could be resolved. Regrettably, due to the limited amount of time available, resolution of the issue could not be accomplished prior to the conclusion of the experiment.

C. Experiment Results

According to the procedure stated in Section VI-A, the reason why each run ended can be classified based on a DUE or an iteration limit (IL). The results of that time window can be classified if any SDC was detected. The total detected faults ware 48 out of 55 from the data set. The resulting analysis yields 4 categories: IL clean, IL+SDC, DUE clean, and DUE+SDC. Figure 7 shows the distribution on a pie chart. The DUE category also includes runs that ended with an unparseable output but were still technically sending characters through the UART (e.g., sending a single repeating character). Among those DUEs, there was only one illegal instruction captured by the exception handler. Around 55% of the runs end in a DUE, while 45% of the runs end on the iteration limit.

1) Analysis of Corner Cases: The system's behaviour under a neutron beam is hard to model but is even harder to automate. Several instances where the SEU did not cause a state that fits the model have been detected. Figure 8 illustrates some of the observed corner cases.

In Figure 8a shows an interesting case, where the 4th bit of each cell got stuck at 0 for all outputs. This is most likely DUE to a PL error. As this turned out to be a stable state, it is deducible that the error did not happen on the instruction critical path. A couple of thousand iterations later, another error happened at the input matrix. Figure 8b shows

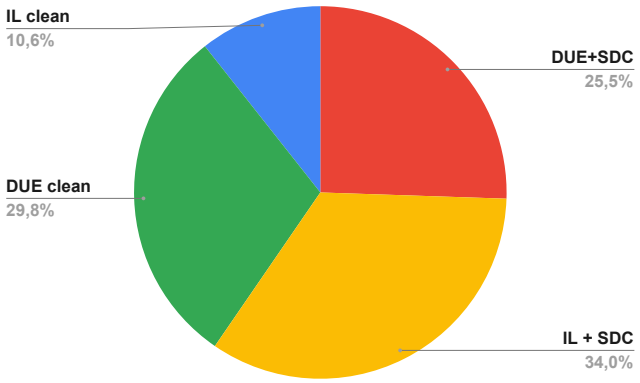
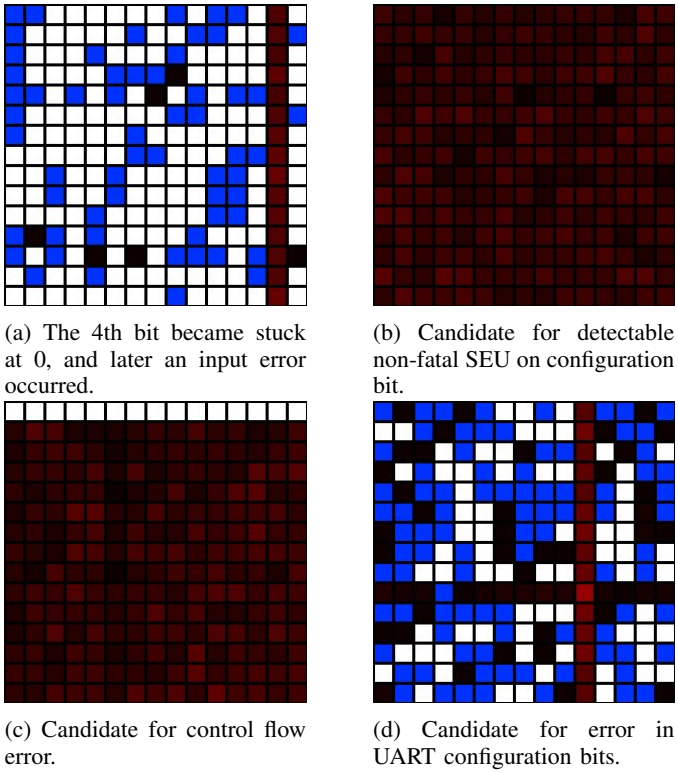


Fig. 7: Distribution of tests between the 4 categories: IL clean, IL+SDC, DUE clean, and DUE+SDC.

a completely wrong matrix. By inspecting each cell reveals that the entire output is composed of the same value, but different from any of the golden results. This did not result in a failure of the processor. On the contrary, it finished the correct number of iterations. As this is a stable result, communication, control flow and input data errors were discarded as all the values are the same. The logical conclusion would be that it is a configuration bit error. The most likely candidate is the computation of the MUL/DIV block, as it would only affect the result of the computation. No other section of the code utilizes these instructions so they would not affect the control flow.

Figure 8c initially looks like a similar case to Figure 8b. However, inspecting the cell values shows that it is a clear control flow issue. The entire output matrix is composed of the values from the first row. Interestingly, this also led to a correct number of iterations and output size. Therefore, it can be concluded that the inner loop of the output dump was skipped entirely. Finally, Figure 8d contains an input error followed by a surge of errors in different locations. Excluding the input error from the analysis, what remains is close to 90% of static errors containing mostly character errors and what appear to be computation errors. Upon further inspection, those remaining errors are in reality skipped or repeated characters. The other 10% are single-bit errors that change positions in a mostly cyclic manner. As the error position cycles persist between function calls, the best candidate for this type of error is the UART buffer of the processor itself. This type of error is common in different variations (e.g. every 8 cells a character is substituted by the character A).

2) Cross Section: Using the approach of the literature and considering the number of SEUs, an overestimation of the cross-section, according to the modeling in Section V, as shown in the *Raw* column of Figure 9, would occur. Instead, if the errors that become a new stable state are filtered first, it results in column *Repeat*. Then, by filtering overcounted errors that can be correctly identified by manual inspection as described in Section VI-C1, a more accurate estimate of the correct number of SEUs that impacted the system shows. At the end of this analysis, from the consecutive iterations



(a) The 4th bit became stuck at 0, and later an input error occurred.

(b) Candidate for detectable non-fatal SEU on configuration bit.

(c) Candidate for control flow error.

(d) Candidate for error in UART configuration bits.

Fig. 8: Different candidates for corner cases of the presented model.

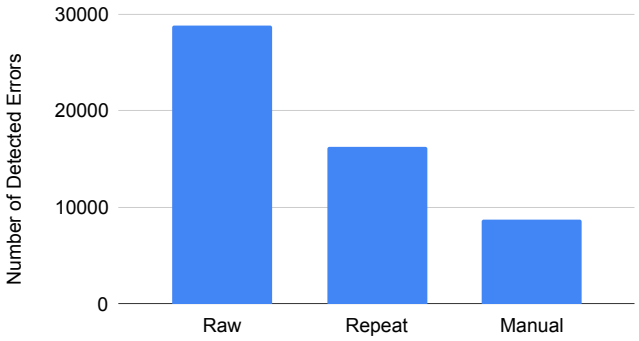


Fig. 9: Error count with different types of filtering.

between each time the device was reprogrammed, half of the permanent errors were caused by input errors, while the rest were caused by CRAM errors or control flow issues. This means that by analyzing the common origin of the errors, the total error count could be reduced by 70%.

With the number of detected errors during the experiment, the *cross-section* can be calculated by using the received particle fluence, as shown in Equation 4. Table I contains the calculated result of the cross-section by taking into account different error counts. Even though the manual approach of laboriously looking at all of the corner cases was necessary due to the diverse ways that the system could fail, the automatic strategy of evaluating changes in permanent states can be used to reduce the number of times reprogramming the device is necessary when testing an SRAM-based FPGAs.

TABLE I: Cross section of the device and chosen benchmark for different levels of filtering.

Filter type	Raw	Repeat	Manual
Total Errors	28.82E+03	16.30E+03	8.72E+03
Cross-Section [cm^2]	3.36E-04	1.90E-04	1.02E-04

D. Observation

Most of the proof is limited to only the results in the output files. The biggest bottleneck is to retrieve the data from the device. The execution of the program was limited by the UART function, as it took significant more time than the calculation itself. This meant that the output results were more exposed to the beam, waiting in the buffer, than the calculation.

VII. Conclusions

In this paper, the performance of the Ibex RISC-V soft core processor has been evaluated through implementation on an SRAM-FPGA, which was exposed to a neutron beam without any protection schemes. The experiment showed that a significant number of errors occurred in the user's memory and during communication steps. An error classification model was used to filter some incorrect outputs, leading to a more accurate estimation of SEUs without requiring continuous reprogramming of the device.

The results of the experiment suggest that most faults are due to SDC in the CRAM, with 55% resulting in DUEs, of which more than half were considered clean. The experiment did not provide refined information on where and how the faults occurred.

In non-critical systems, full error coverage may not be worth the trade-off in terms of performance, area, power, or development costs. Hence, reliability techniques should be applied only to the components that are most likely to generate a detectable error.

The simulation results provide insight into the symptoms caused by fault injection, making it possible to understand why certain fault protection schemes might be used. However, the results of the experiment are different from the simulation, highlighting the limitations of observing the hardware in a simulated environment. In simulation, the hardware can be monitored with complete visibility, but this can create a blind spot for the practical problems that arise during data sharing between the DUT and the observer.

To address the limitations of the experiment, a complementary strategy and necessary toolset should be developed to improve the physical setup and focus on the bottleneck. The next step should be to test a flash-based FPGA instead of a SRAM-based one. This might be a significant improvement compared to the SRAM FPGA only based on memory technology. The question of observability remains unanswered for now, and modules such as FIRECAP, which act as a watchdog and recovery tool, may offer a solution in the future

Appendix A
Acronyms

BR Background Radiation. 2

BSD Berkeley Software Distribution. 3

CISC Complex instruction set computer. 3
COTS Commercially off-the-shelf. 1, 2
CRAM Configuration RAM. 7, 9

DIVA Dynamic Implementation Verification Architecture. 5
DMR Dual Modular Redundancy. 4
DRAM Dynamic Random Access Memory. 4
DUE Detected Unrecoverable Error. 7–10
DUT Device under test. 1, 7, 8, 10

ECC Error Correcting Code. 1–4
ESA European Space Agency. 2

FPGA Field Programable Gate Array. 1, 2, 5, 7, 8, 10

HD Hamming Distance. 6, 7

IC Integrated Circuit. 2
IP Interlactual Property. 3, 5, 8
ISA Instruction Set Architecture. 1, 3

MBU Mutiple Bit Upsets. 3, 5

PMP Physical Memory Protection. 1, 4

RISC Reduced instruction set computer. 3
ROM Read-Only Memory. 4
RVFI RISC-V Formal Interface. 6

SBU Single Bit Upsets. 5
SDC Silent Data Coruption. 2, 3, 5, 7–10
SEE Single Event Effects. 1–5
SEM Soft Error Mitigation. 5, 8
SET Single Event Transient. 2
SEU Single Event Upset. 2–4, 6–10
SRAM Static Random Access Memory. 4

TMR Triple Modular Redundancy. 1–5

Acknowledgment

First, I would like to thank Bruno Forlin for all his effort and support in too my project. Second, I would like to tank Marco Ottavi for the knowledge and the opportunity I got from him. Without these two persons I would not have manged to come to a conclusion of my master. And naturally, without the loving support of my parents, Annemiek and Chris, this howl endeavor was not possible.

"*Is er leven op Pluto*
Kun je dansen op de maan
Is er een plaats tussen de sterren
Waar ik heen kan gaan" [16]

Space, the last frontier. How many of us look up to the sky and ask our self what is up there? Like the famous Dutch song [16] is there a place between the stars where I could go? Maybe getting buried on the Moon is most feasible for me. But in order to get there we need to be able to depend on

our technic, because space is a hazardous environment to us humans and machine. That is what compelled me to this topic, the combination of a dream and a question of dependability. For those whom are wondering, my background is in robotics and especially now in the brain box of the robot. I see a future of autonomous robots that explore the universe. Luckily, my job will be one of the last to be automated. For now, let us get back to earth. Progress goes quick, but not that quick.

Wouter van Huffelen
Enschede, 28-02-2023

REFERENCES

[1] Muhammad Ali Akbar, Bo Wang, and Amine Bermak. "Self-Repairing Hybrid Adder With Hot-Standby Topology Using Fault-Localization". In: *IEEE Access* 8 (2020), pp. 150051–150058. DOI: 10.1109/ACCESS.2020.3016427.

[2] Douglas Almeida dos Santos et al. "Characterization of a RISC-V System-on-Chip under Neutron Radiation". In: *DTIS 2021 - 16th International Conference on Design Technology of Integrated Systems in Nanoscale Era*. Montpellier, France, June 2021. DOI: 10.1109/DTIS53253.2021.9505054. URL: https://hal-lirmm.ccsd.cnrs.fr/lirmm-03357515.

[3] Jan Andersson. "Development of a NOEL-V RISC-V SoC Targeting Space Applications". In: *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. 2020, pp. 66–67. DOI: 10.1109/DSN-W50199.2020.00020.

[4] T.M. Austin. "DIVA: a reliable substrate for deep submicron microarchitecture design". In: *MICRO-32. Proceedings of the 32nd Annual ACM/IEEE International Symposium on Microarchitecture*. 1999, pp. 196–207. DOI: 10.1109/MICRO.1999.809458.

[5] T. Bates and C. P. Bridges. "Single event mitigation for Xilinx 7-series FPGAs". In: *2018 IEEE Aerospace Conference*. 2018, pp. 1–12. DOI: 10.1109/AERO.2018.8396520.

[6] Gavin Burtt. *How An Ionizing Particle From Outer Space Helped A Mario Speedrunner Save Time*. URL: https://www.thegamer.com/how-ionizing-particle-outer-space-helped-super-mario-64-speedrunner-save-time/.

[7] 2022 CERN. *Cosmic rays: particles from outer space — CERN*. URL: https://home.cern/science/physics/cosmic-rays-particles-outer-space.

[8] Greg Chadwick and Marno van der Maas. *Ibex Super System*. URL: https://github.com/GregAC/ibex_super_system.

[9] Abdallah Cheikh et al. "Klessydra-T: Designing Vector Coprocessors for Multithreaded Edge-Computing Cores". In: *IEEE Micro* 41.2 (2021), pp. 64–71. DOI: 10.1109/MM.2021.3050962.

[10] Nan Chen et al. "MSRP-FT: Reliable Resource Sharing on Multiprocessor Mixed-Criticality Systems". In: *2022 IEEE 28th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 2022, pp. 201–213. DOI: 10.1109/RTAS54340.2022.00024.

[11] Pasquale Davide Schiavone et al. "Slow and steady wins the race? A comparison of ultra-low-power RISC-V cores for Internet-of-Things applications". In: *2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*. 2017, pp. 1–8. DOI: 10.1109/PATMOS.2017.8106976.

[12] Stefano Di Mascio et al. "The Case for RISC-V in Space". In: *Applications in Electronics Pervading Industry, Environment and Society*. Ed. by Sergio Saponara and Alessandro De Gloria. Cham: Springer International Publishing, 2019, pp. 319–325. ISBN: 978-3-030-11973-7.

[13] Fernando Fernandes dos Santos, Angeliki Kritikakou, and Olivier Sentieys. "Experimental evaluation of neutron-induced errors on a multicore RISC-V platform". In: *IOLTS 2022 - 28th IEEE International Symposium on OnLine Testing and Robust System Design*. Paper accepted on 28th IEEE IOLTS 2022. Torino, Italy: IEEE, Sept. 2022, pp. 1–7. URL: https://hal.inria.fr/hal-03697265.

[14] Wikipedia foundation. *Abraham Wald*. URL: https://en.wikipedia.org/wiki/Abraham_Wald.

[15] Michael Gautschi et al. "Near-Threshold RISC-V Core With DSP Extensions for Scalable IoT Endpoint Devices". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 25.10 (2017), pp. 2700–2713. DOI: 10.1109/TVLSI.2017.2654506.

[16] Henk Temming Henk Westbroek. *België*. 1982.

[17] A. Steve Johnson et al. *Spaceflight Radiation Health Program at JSC*. URL: https://www.veripool.org/verilator/.

[18] Olof Kindgren. "Invited Paper: A Scalable Approach to IP Management with FuseSoC". In: (2019). DOI: https://osda.gitlab.io/19/kindgren.pdf.

[19] Paul Kocher et al. "Spectre Attacks: Exploiting Speculative Execution". In: *40th IEEE Symposium on Security and Privacy (SP'19)*. 2019.

[20] JunLin Li et al. "Study on Transient Ionizing Radiation Effect of 40nm SRAM". In: *2019 3rd International Conference on Radiation Effects of Electronic Devices (ICREED)*. 2019, pp. 1–4. DOI: 10.1109/ICREED49760.2019.9205175.

[21] A. Lindoso et al. "A Hybrid Fault-Tolerant LEON3 Soft Core Processor Implemented in Low-End SRAM FPGA". In: *IEEE Transactions on Nuclear Science* 64.1 (2017), pp. 374–381. DOI: 10.1109/TNS.2016.2636574.

[22] Moritz Lipp et al. "Meltdown: Reading Kernel Memory from User Space". In: *27th USENIX Security Symposium (USENIX Security 18)*. 2018.

[23] Pierre Maillard et al. "Single-Event Upsets Characterization Evaluation of Xilinx UltraScale™ Soft Error Mitigation (SEM IP) Tool". In: *2016 IEEE Radiation*

Effects Data Workshop (REDW). 2016, pp. 1–4. DOI: 10.1109/NSREC.2016.7891745.

[24] Albert Meixner, Michael E. Bauer, and Daniel Sorin. "Argus: Low-Cost, Comprehensive Error Detection in Simple Cores". In: *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*. 2007, pp. 210–222. DOI: 10.1109/MICRO.2007.18.

[25] Albert Meixner and Daniel J. Sorin. "Error Detection Using Dynamic Dataflow Verification". In: *16th International Conference on Parallel Architecture and Compilation Techniques (PACT 2007)*. 2007, pp. 104–118. DOI: 10.1109/PACT.2007.4336204.

[26] M.V. O'Bryan et al. "Current single event effects and radiation damage results for candidate spacecraft electronics". In: *IEEE Radiation Effects Data Workshop*. 2002, pp. 82–105. DOI: 10.1109/REDW.2002.1045537.

[27] Daniel A. G. Oliveira et al. "GPGPUs ECC efficiency and efficacy". In: *2014 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. 2014, pp. 209–215. DOI: 10.1109/DFT.2014.6962085.

[28] Andrey Petrov et al. "Effects of space radiation on resistive memory and comparison with other types of non-volatile memory". In: *2022 Moscow Workshop on Electronic and Networking Technologies (MWENT)*. 2022, pp. 1–4. DOI: 10.1109/MWENT55238.2022.9802312.

[29] Heather Quinn et al. "Software Resilience and the Effectiveness of Software Mitigation in Microcontrollers". In: *IEEE Transactions on Nuclear Science* 62.6 (2015), pp. 2532–2538. DOI: 10.1109/TNS.2015.2496342.

[30] 27-Jun-2022 RISC. *History - RISC-V international*. URL: https://riscv.org/about/history/.

[31] Douglas Almeida Santos et al. "A Low-Cost Fault-Tolerant RISC-V Processor for Space Systems". In: *2020 15th Design Technology of Integrated Systems in Nanoscale Era (DTIS)*. 2020, pp. 1–5. DOI: 10.1109/DTIS48698.2020.9081185.

[32] Wilson Snyder. *Verilator*. URL: https://srag.jsc.nasa.gov/Publications/TM104782/techmemo.htm.

[33] Sébastien Thomet et al. "FIRECAP: Fail-Reason Capturing hardware module for a RISC-V based System on a Chip". In: *2021 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. 2021, pp. 1–6. DOI: 10.1109/DFT52944.2021.9568317.

[34] Imran Wali et al. "Analyzing the impact of the Operating System on the Reliability of a RISC-V FPGA Implementation". In: *2020 27th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. 2020, pp. 1–4. DOI: 10.1109/ICECS49266.2020.9294858.

[35] Nils-Johan Wessman et al. "De-RISC: the First RISC-V Space-Grade Platform for Safety-Critical Systems". In: *2021 IEEE Space Computing Conference (SCC)*. 2021, pp. 17–26. DOI: 10.1109/SCC49971.2021.00010.

[36] Andrew E. Wilson et al. "Neutron Radiation Testing of a TMR VexRiscv Soft Processor on SRAM-Based FPGAs". In: *IEEE Transactions on Nuclear Science* 68.5 (2021), pp. 1054–1060. DOI: 10.1109/TNS.2021.3068835.

[37] Xilinx. "RT Kintex UltraScale FPGAs for Ultra High Throughput and High Bandwidth Applications". In: (2020). URL: https://docs.xilinx.com/v/u/en-US/wp523-xqrku060.