FPGA-BASED FAULT INJECTOR FOR SEU-ROBUSTNESS ANALYSIS OF SCOSA

F. Brömer*†, P. Kenny‡, A. Lund‡, C. Gremzow†, D. Lüdtke*

* Institute for Software Technology, German Aerospace Center (DLR), Lilienthalplatz 7, 38108 Braunschweig, Germany

† University of Applied Sciences (HTW) Berlin, Wilhelminenhofstr. 75A, 12459 Berlin, Germany

‡ Institute for Software Technology, German Aerospace Center (DLR), Münchener Str. 20, 82234 Weßling, Germany

Abstract

The Scalable On-board Computer for Space Avionics (ScOSA) project aims to develop an on-board computer which offers both reliability and high-performance through the use of a heterogeneous distributed system of commercial-off-the-shelf and radiation-hardened processors. This system should operate without failures even in the presence of single-event upsets (SEUs), which are common occurrences for electronic systems in space. The ScOSA middleware includes several fault detection, isolation and recovery (FDIR) mechanisms for coping with faults, but their effectiveness in the presence of radiation has not yet been proven, as testing such effects on the ground is challenging. This paper presents our approach to investigate the effect of single-event upsets on the ScOSA system and the effectiveness of its error handling mechanisms in their presence. A fault injector has been instantiated in the FPGA co-processor of a commercial-off-the-shelf Xilinx system-on-chip from the Zynq 7000 family using a Microblaze soft processor, which is used to simulate the effect of SEUs by flipping bits in the main memory used by the kernel, middleware and applications.
A machine-learning-based image processing algorithm will be used as an example application and run using the ScOSA middleware while the fault injector is active. The system will be executed multiple times, with faults injected into different memory locations and at different times in each run. The system will be monitored for FDIR events and unrecoverable failures. The operation of the middleware and the results of the sample application will be compared to the results of a golden run, where no faults are injected, to assess the number of unhandled errors at the middleware and application levels. The results are classified by severity, such as incorrect algorithm results, handled FDIR events and unhandled system crashes. These results will then be correlated with the fault location, such as kernel or application memory. By applying SEU simulation techniques to an on-board software system, we aim to demonstrate the usefulness of such simulations as well as guiding the further development of the ScOSA system to target further SEU mitigation efforts and improve the system's robustness, as well as characterizing the system's robustness to SEUs occurring in different locations.

Keywords

COTS, SEU, FPGA, fault tolerance, reliable computing

1. INTRODUCTION

A big challenge for electronic devices in space is the high amount of radiation they are exposed to. The radiation induces so-called single-event effects (SEEs) in the integrated circuits. SEEs can result in permanent damage of electronic components, e.g. single-event latchups, or to temporary effects, the so-called soft errors or single-event upsets (SEUs). Usually engineers tend to either shield the electronic devices or implement redundancy when a high-radiation environment is expected. Furthermore, radiation-hardened components can be used, which are designed for such an environment and already implement redundancy, shielding or technologies which are less affected by radiation. All of those strategies eventually lead to increased costs, especially in the context of space missions. For this reason, nowadays missions tend to implement low-cost commercial-off-the-shelf (COTS) hardware combined with fault-tolerant software, which is able to cope with a certain number of errors.

The overall use of COTS hardware in space missions has been increasing in the last decade. Especially for FPGAs, space-grade hardware is significantly more expensive [1], while also lacking performance compared to modern COTS devices [2]. However, the difference in reliability and error resilience often justifies the usage of space-grade hardware.

doi: 10.25967/570406

To compensate for the lack of reliability in COTS devices and enable the use of its improved performance, different error detection mechanisms for various devices have been tested, in space environments as well as in other radiation environments [3]. To test these error handling mechanisms, this paper presents an FPGA-based fault injector and the outlook on how to use it to analyze the robustness of a system and its susceptibility to the effects of radiation events.

The fault injector is intended to be used in a flight experiment of the the German Aerospace Center (DLR), namely the ScOSA Flight Experiment. It will be used to test the integrated FDIR functionality of the ScOSA system, as well as its SEU detection. To achieve that, it will be run while an example application is executed, to test the recovery mechanisms in a realistic setting. The example program is a machine-learning image processing application, which will also be used in the flight experiment.

2. RELATED WORK

The German Aerospace Centre (DLR) researches in the field of SEU robustness by means of a flight project in which a new on-board computer architecture shall be evaluated. It is called Scalable On-board Computer for Space Avionics (ScOSA Flight Experiment) [2] based on the projects ScOSA [4] and its predecessor OBC-NG [5]. While the earlier projects resulted in a reliable on-board architecture, the ScOSA flight experiment aims to further develop the on-board system. Furthermore, the overall goal of the flight experiment project is to increase the technical readiness level of the on-board computer by means of an extensive evaluation in-orbit. For this reason, several space applications shall be executed by ScOSA. One of them will be the SEU detection, which will be developed with the help of the SEU injector. In this way, we can compare the results from the SEU detector in-orbit with the results produced on ground using the SEU injector in order to understand how realistic the fault injection is.

The ScOSA system is a distributed and heterogeneous hardware architecture combined with a layered middleware utilizing the Tasking Framework [6] as an execution platform. The central idea of ScOSA is that it combines radiation-hardened processors with COTS processors, such that it can take advantage of the reliability of the former and the performance of the latter. The ScOSA middleware abstracts this distributed architecture away from the application developer and at the same time it utilizes it to recover from soft errors which led to processor crashes. In the presence of a processor loss, the middleware will automatically activate a new configuration in which the tasks that were executed by the lost processor will migrate to remaining processors. In addition to this reconfiguration mechanism the middleware provides the application developer with common tools to increase the fault-tolerance, such as checkpointing states and triple modular redundancy (TMR). Furthermore, the middleware comes with a network protocol which supports Ethernet and SpaceWire.

The protocol enables the middleware to send messages reliably to other participants of the distributed system. Application developers will implement their space application by means of the Tasking Framework, such that the middleware can handle the applications for reconfiguration. The Tasking Framework decomposes applications into tasks, which are stateless, and channels which connect the tasks to each other and contain data. With this data-flow oriented programming paradigm, the middleware is able to allocate the tasks to different nodes and reorganize this allocation in case of failed nodes.

As space systems need to meet high requirements regarding reliability and robustness, testing is an important part of the development of such systems. Generally, SEUs occur under the influence of heavy ion and proton radiation and are difficult to predict. Since generating that kind of radiation is complicated and expensive, it is not common to generate them on-ground for testing. Instead, different types of simulation have been applied to predict the behaviour of electronic components. One possible example for a solution is the fully-physical simulation based on measured in-orbit data [7].

However, to analyze the behaviour in error cases, fault injection has proven to be useful and FPGA-based fault injectors have been successfully developed, for example in the *SCHIFI*-project [8], which focusses on the development of a highly flexible fault injector. However, its fault injection does not happen during runtime, but before the program execution, which is an important difference to the fault injector presented in this paper.

Another example of FPGA-based fault injection demonstrates the usefulness of this technology in the area of reliability testing, as the work by Miklo et al. is not only capable of real-time fault injection but can also be used in safety-critical areas such as nuclear power applications [9].

The machine-based image classification application that will be used as an example application for the developed fault injector is currently being developed. It is based on the OBPMark project [10], which combines machine-learning object detection with benchmarking of on-board software in spacecraft.

3. THEORETICAL BACKGROUND

SEEs are radiation events caused by radiation in the upper atmosphere or space environments. They occur when a charged particle with high energy hits a

semiconductor circuit and interferes with the electrical behaviour of the circuit [11].
They can lead to different kinds of errors, ranging from single-bit changes to permanent damage to the circuit. Even though a bitflip does not cause damage on a hardware level, the consequences can still be severe. Ranging from a changed value to wrong control sequences, the occurrence of a soft error can even cause a mission loss if not adequately handled.

Even though a hardware-level simulation of SEUs is possible and simulates the issues on a level that makes it possible to make statements about the behaviour of the circuit, it is not necessary for testing the functionality. For testing or validating the design, the complexity of such a simulation might even be a disadvantage. An alternative way to simulate the behaviour of the system, without adding the complexity of particle-level simulations, is fault injection. While it doesn't provide the randomness or accuracy of a fully-physical simulation, it provides the ability to put the system into a state equivalent that following a soft error occurrence.

The usage of fault injection also has some advantages from the perspective of testing. It enables us to generate faults in specific areas of the circuit, manipulate the distribution in time and memory of the occurring faults and repeat the test when the system is in various states. Moreover, the rates with which radiation errors occur in reality are quite low [12] and depend highly on the environment [13], which would make testing under real-life circumstances even harder.

In our work we evaluate the fault injection capabilities of a Xilinx Zynq 7020 system-on-chip as a hardware platform. It is a commonly used COTS hardware platform which consists of an Arm Cortex A9 processing system and an FPGA fabricated in a 28nm process. It provides a sufficient amount of processing power and also a variety of interfaces to the processing system as well as peripherals.

4. FAULT INJECTION

The basic idea of the project is to use the memory interfaces of the FPGA to manipulate contents located in the processing system's memory. This is achieved by using the Advanced eXtensible Interface (AXI) bus of the programmable logic (PL) part of the SoC.

The Zynq 7020 platform provides different interfaces between the PL and the memory. For using the AXI interface, Xilinx provides two types of interfaces. While the accelerator ports utilize caches for acceleration, the high-performance ports provide an uncached access to the memory. For the purpose of direct memory access, the high-performance ports are preferred. To use those, the interface needs to be accessed by the PL. An AXI interconnect module, provided as an intellectual property (IP) block from the FPGA manufacturer Xilinx, was used. The IP block is available in the IP catalogue integrated into the Xilinx development tool Vivado and can be used with every Xilinx hardware that supports the AXI infrastructure.

To control the memory interface and manipulate specific hardware addresses, the fault injector contains a softcore processor which is instantiated in the FPGA. While any simple processor would be suitable, a Xilinx Microblaze is used because it is known to work well with the other components from the manufacturer. The block diagram representing the FPGA design is shown in Figure 1. The processor is programmed bare-metal, as it the functionality and complexity of an operating system is not needed. The number of injected errors can be configured in software, as well as the distribution in time and memory. To control the Microblaze software, an external UART interface is used to transmit control commands that were defined beforehand. Implementing the control commands with an external interface makes the fault injector independent from the system under test and eliminates the possibility of interfering with its own control instance. Furthermore, implementing the fault injector as an FPGA component instead of a software solution has several advantages. On a hardware level, the fault injector does not interfere with the timing of the system under test and does not add extra clock cycles to avoid affecting the overall timing of the circuit.

The Microblaze is programmed in C using the Vitis programming platform provided by Xilinx. The memory areas for the fault injection can be addressed directly using their hardware addresses, since they are known from the address map defined by the AXI interface. On the side of the processing system, the Linux operating system will allocate memory to the middleware and applications at runtime. These are virtual addresses that can be resolved to their physical counterparts using Linux system calls.

The program itself implements functions to flip specific bits at a given address, as well as flipping a given number of bits randomly within an address range defined by the control commands. To make the random distribution of injected faults reproduceable, the random-number generator produces pseudo-random locations based on a seed value that is also communicated using a control command.

Even though the Microblaze processor has more capabilities than what is currently used, the hardware implementation only uses a small part of the FPGA resources which enables the SoC to use the PL part for other components as well. As shown in Table 1, the utilization after completing synthesis and implementation uses up to 10% of the FPGA resources. It

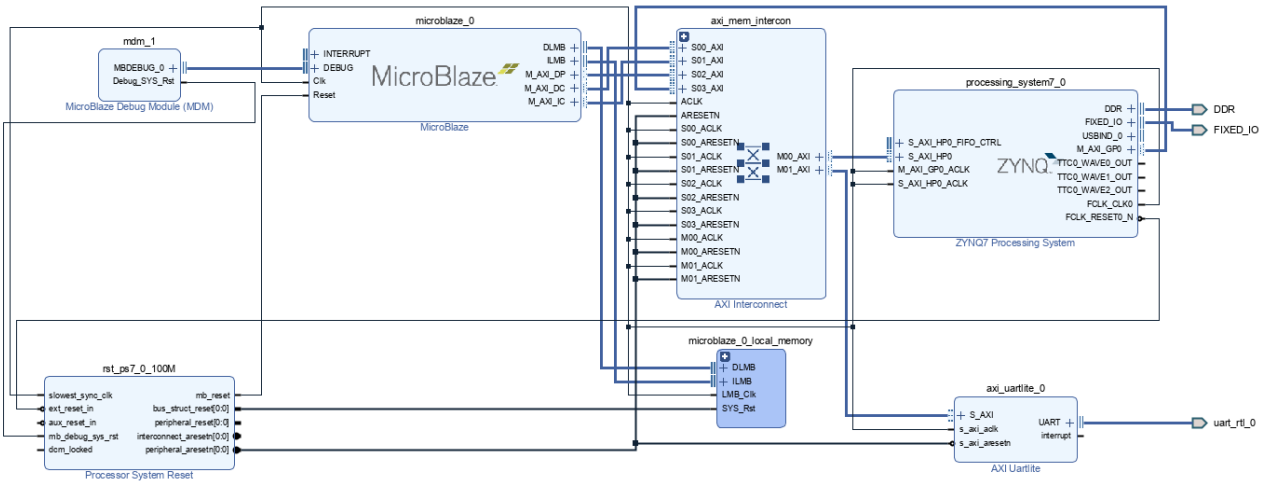

FIG 1. Block diagram of the hardware components for fault injection

also shows the absolute utilization, broken down by different resources available in the FPGA.

Resource	Utilization	Available	Utilization %
LUT	3992	53200	7.50
LUTRAM	462	17400	2.66
FF	4333	106400	4.07
BRAM	14	140	10.00
IO	2	200	1.00
BUFG	3	32	9.38

TAB 1. FPGA utilization for different resources after implementation

5. FAULT DETECTION

A simple method for detecting SEUs in memory is to continually read the memory using the CPU and monitor its contents for changes. Using this method, the SEU detector requests an area in main memory with non-cached access from the operating system, obtains its physical address and passes this address to the programmable logic. The detector initializes this memory by filling it with a known bit pattern and continually reads back the stored values, monitoring for any deviation from the known pattern. This method allows the detector to reliably detect any changes in this memory caused by SEUs, or in validation, by the fault injector. However, it is only applicable for memory areas which are dedicated exclusively to SEU detection. This makes it useful for characterizing the rate of SEU occurrence in main memory in flight for particular hardware, but it cannot be used as part of an FDIR system to detect and subsequently mitigate SEUs occurring in memory used for other purposes.

In order to detect SEUs occurring in memory used by other parts of the system, the ScOSA middleware implements higher-level FDIR mechanisms [2]. These include heartbeat messages sent between nodes to allow monitoring of the status by other nodes; a reconfiguration service to allow applications to migrate to other nodes if their host node fails; a reintegration service to allow restarted nodes to rejoin the system; checkpoints which back up applications' state data to the local and remote nodes to allow recovery after restarts; and a voter service, providing triple-modular redundancy. In order to maximize the amount of memory available to applications, the error-correcting codes in the memory are not enabled. Instead, ScOSA relies on its own FDIR mechanisms. Our future work will include analyzing the effectiveness of these mechanisms, using the FPGA-based fault injector to simulate SEUs in the memory.

The faults caused by SEUs can lead to failures at several severity levels. If an SEU occurs in memory which is then overwritten before any further read operations on that memory take place, the fault is effectively scrubbed and will not propagate to a failure. If the memory is read, the program execution could remain correct but lead to incorrect algorithm results. It could also lead to incorrect program execution resulting in a crash of a single process or, especially if the error occurs in the memory of the operating system, a crash of the entire node. We categorize these failures as incorrect algorithm results, handled FDIR events and unhandled system crashes respectively, in a method similar to that used by Carlisle and George [14].

To detect the first of these failures, incorrect algorithm results, we intend to run an image-processing machine-learning algorithm, representative of a typical on-board processing application, both with and without fault injection. By running the algorithm without fault injection, we generate a "golden run" whose results can be compared to subsequent runs with active fault injection. Following completion of the algorithm, the results will be compared offline

to those of the golden run. If the algorithm runs to completion and no faults are detected by the FDIR but the offline analysis finds a difference to the results of the golden run, we categorize the failure as an incorrect algorithm result. If no difference to the golden run is found, it is categorized as no failure.

In order to assess the effectiveness of the ScOSA middleware's inbuilt FDIR mechanisms, we monitor the notifications output by the middleware to detect anomalous behavior and FDIR activity. If FDIR actions are necessary, such as the restarting of a node, but the algorithm results are identical to those of the golden run, we characterize the run as a handled FDIR event.

In the most serious case, an SEU or series of SEUs may lead to system crashes that the FDIR system cannot handle or does not handle correctly. We characterize the run as an unhandled system crash if this results in no output being produced, or the output differing from the golden run despite FDIR intervention.

By injecting SEUs into different parts of the system, such as kernel memory, middleware memory, application memory and FPGA registers, and characterizing the results, we aim to demonstrate the effectiveness of ScOSA's FDIR mechanisms, and also identify weaknesses, which can then be used to improve the FDIR mechanisms and assess the risk for different mission requirements.

6. SUMMARY & OUTLOOK

This paper gave an overview of the FPGA-based fault injection implemented for the ScOSA project and the related SEU detection mechanisms. It showed how a hardware design based on a Microblaze processor can be used to manipulate data located in the memory of a Zynq 7020 platform and how it can be controlled.

The next goals are integrating the fault injector with the ScOSA system and its SEU detection. To test the ScOSA middleware under realistic conditions, the test runs will be executed with an example application running. The chosen application is a machine-learning-based image-classification algorithm which will be run using the ScOSA middleware while the fault injector is active. The system will be executed multiple times, with faults injected into different locations and at different times in each run. The distribution of the injected errors can be either pseudo-randomly determined or controlled manually, which applies to both the distribution in time and memory. That means that the faults can also be injected in specific memory areas, such as those used by the operating system, the middleware or the application. To make the test runs reproduceable, the pseudo-random runs are created using a seed value for the random number generator which will be stored together with the results of the corresponding run.

The system will be monitored for FDIR events and unrecoverable failures. An unrecoverable failure occurs when the monitored system is in a non-functional state without the ability to recover from it by itself, using the implemented FDIR mechanisms. The occurrence of an unrecoverable failure would also detect possible weaknesses in the ScOSA FDIR mechanisms.

The operation of the middleware and the results of the sample application will be compared to the results of a golden run, where no faults are injected, to assess the number of unhandled errors at the middleware and application levels. The results are classified by severity, such as incorrect algorithm results, handled FDIR events and unhandled system crashes. These results will then be correlated with the fault location, such as kernel memory or application memory. The fault injector also serves the purpose of testing the SEU detection, as it will be part of the ScOSA flight experiment.

By applying SEU simulation techniques to an on-board software system, we aim to demonstrate the usefulness of such simulations as well as guiding the further development of the ScOSA system to target further SEU mitigation efforts and improve the system's robustness, as well as characterizing it to SEUs occurring in different locations.

Contact address:

fiona.broemer@dlr.de

References

[1] Harshad Bokil. COTS Semiconductor Components for the New Space Industry. In *2020 4th IEEE Electron Devices Technology and Manufacturing Conference (EDTM)*, pages 1–4, 2020. DOI: 10.1109/EDTM47692.2020.9117834.

[2] A. Lund, Z.A. Haj Hammadeh, P. Kenny, V. Vishav, Andrii Kovalov, Hannes Watolla, Andreas Gerndt, and Daniel Lüdtke. ScOSA system software: the reliable and scalable middleware for a heterogeneous and distributed on-board computer architecture. In *2022 CEAS Space Journal*, pages 161–171, 2022. DOI: https://doi.org/10.1007/s12567-021-00371-7.

[3] R. Giordano, A. Aloisio, S. Massarotti, G. Tortone, Y.-T. Lai, S. Korpar, R. Pestotnik, L. Santelj, A. Lozar, M. Shoji, and S. Nishida. On-the-Fly Self-Reconfiguring FPGAs for Single Event Upset Monitoring at Belle II. In *2020 IEEE Nuclear Science Symposium and Medical Imag-*

ing Conference (NSS/MIC), pages 1–3, 2020. DOI: 10.1109/NSS/MIC42677.2020.9507982.

[4] Carl Johann Treudler, Heike Benninghoff, Kai Borchers, Bernhard Brunner, Jan Cremer, Michael Dumke, Thomas Gärtner, Kilian Johann Höflinger, Daniel Lüdtke, Ting Peng, Eicke-Alexander Risse, Kurt Schwenk, Martin Stelzer, Moritz Ulmer, Simon Vellas, and Karsten Westerdorff. ScOSA - Scalable On-Board Computing for Space Avionics. In *IAC 2018*, Oktober 2018.

[5] Daniel Lüdtke, Karsten Westerdorff, Kai Stohlmann, Anko Börner, Olaf Maibaum, Ting Peng, Benjamin Weps, Görschwin Fey, and Andreas Gerndt. OBC-NG: Towards a reconfigurable on-board computing architecture for spacecraft. In *2014 IEEE Aerospace Conference*, pages 1–13, 2014. DOI: 10.1109/AERO.2014.6836179.

[6] Zain A. H. Hammadeh, Tobias Franz, Olaf Maibaum, Andreas Gerndt, and Daniel Lüdtke. Event-Driven Multithreading Execution Platform for Real-Time On-Board Software Systems. In *15th annual workshop on Operating Systems Platforms for Embedded Real-Time applications*, pages 29–34, Juli 2019.

[7] Yi Sun, Hong-Wei Zhang, Zhi-Chao Wei, Qing-Kui Yu, Min Tang, Chen Shen, and Ding Gong. Heavy ion-and Proton-induced SEU Simulation and Error Rates Calculation in 0.15um SRAM-based FPGA. In *2019 3rd International Conference on Circuits, System and Simulation (ICCSS)*, pages 84–88, 2019. DOI: 10.1109/CIRSYSSIM.2019.8935625.

[8] Suman Sau, Maha Kooli, Giorgio Di Natale, Alberto Bosio, and Amlan Chakrabarti. SCHIFI: Scalable and flexible high performance FPGA-based fault injector. In *2016 Conference on Design of Circuits and Integrated Systems (DCIS)*, pages 1–4, 2016. DOI: 10.1109/DCIS.2016.7845375.

[9] Marko Miklo, Carl R. Elks, and Ronald D. Williams. Design of a high performance FPGA based fault injector for real-time safety-critical systems. In *ASAP 2011 - 22nd IEEE International Conference on Application-specific Systems, Architectures and Processors*, pages 243–246, 2011. DOI: 10.1109/ASAP.2011.6043278.

[10] David Steenari, Leonidas Kosmidis, Ivan Rodriquez, Alvaro Jover, and Kyra Förster. OBPMark (On-Board Processing Benchmarks)-Open Source Computational Performance Benchmarks for Space Applications. In *ESA/DLR/CNES European Workshop on On-Board Data Processing (OBDP)*, 2021.

[11] Kalle Ngo, Tage Mohammadat, and Johnny Öberg. Towards a single event upset detector based on COTS FPGA. In *2017 IEEE Nordic Circuits and Systems Conference (NORCAS): NORCHIP and International Symposium of System-on-Chip (SoC)*, pages 1–6, 2017. DOI: 10.1109/NORCHIP.2017.8124960.

[12] Qingyu Chen, Li Chen, David M. Hiemstra, and Valeri Kirischian. Single Event Upset Characterization of the Cyclone V Field Programmable Gate Array Using Proton Irradiation. In *2019 IEEE Radiation Effects Data Workshop*, pages 1–5, 2019. DOI: 10.1109/REDW.2019.8906630.

[13] Kirolosse M. Girgis, Tohru Hada, and Shuichi Matsukiyo. Estimation of Single Event Upset (SEU) rates inside the SAA during the geomagnetic storm event of 15 May 2005. In *2021 IEEE International Conference on Wireless for Space and Extreme Environments (WiSEE)*, pages 27–30, 2021. DOI: 10.1109/WiSEE50203.2021.9613828.

[14] Edward Carlisle, Nicholas Wulf, James MacKinnon, and Alan George. DrSEUs: A dynamic robust single-event upset simulator. In *2016 IEEE Aerospace Conference*, pages 1–11, 2016. DOI: 10.1109/AERO.2016.7500787.