Processor Checkpoint Recovery for Transient Faults in Critical Applications

Paulo R. C. Villa*, Rodrigo Travessini*, Fabian L. Vargas†, Eduardo A. Bezerra*‡
* Electrical Engineering Department, Federal University of Santa Catarina - UFSC, Brazil.
paulo.villa@eel.ufsc.br, rodrigo.travessini@eel.ufsc.br, eduardo.bezerra@ufsc.br
† Electrical Engineering Department, Catholic University - PUCRS, Brazil. vargas@computer.org
‡ LIRMM - Université de Montpellier, France - eduardo.bezerra@lirmm.fr

Abstract—**Electronic systems devoted to critical applications that have to operate in harsh environments, such as the ionizing radiation and electromagnetic interference ones, are very susceptible to soft errors, leading the system to an unwanted state. The most adopted solution to deal with this problem consists in the use of spatial redundancy as an alternative to temporal redundancy. This paper describes an approach to implement checkpoint recovery on FPGA with application-level transparency. The proposed approach was implemented in the LEON3 softcore processor in order to be validated. The case study demonstrates that it is possible to keep data consistency after error injection by performing a process of checkpoint and recovery at runtime, during the execution of a set of applications.**

Keywords—Fault Tolerance, FPGA Reliability, Checkpoint Recovery, Radiation.

I. Introduction

When considering the ever-shrinking dimensions of today's manufacturing technologies, electronic systems have become susceptible to cosmic radiation even at the sea level [1]. These effects can cause temporary or permanent failures leading the system to an unwanted state. Therefore, soft-error is a major concern for dependability of integrated circuits [2]. Strategies to deal with these problems are well known and redundancy is the most used one. Examples of redundancy indlude: spatial redundancy, e.g. *Triple Modular Redundancy* (TMR) [3]; and temporal redundancy, e.g. *Checkpoint Recovery* (CR) [4].

Opposed to the more common spatial redundancy, which uses multiple processors to perform the same computations with a voter logic, the temporal redundancy consists of using only one processor to run twice the same computation, followed by a comparison of the results. This approach allows the detection of inconsistencies between the executions due to unwanted bit-flips.

Space applications have used soft-processors embedded in *Field Programmable Gate Array*s (FPGAs) in several missions [5, 6, 7, 8], but always taking into account the fault tolerance, due to the great influence of radiation and its effects in integrated circuits [9, 10]. In these circumstances, finding a trade-off between processing capacity and reliability level of processors is an important research problem.

In some space programmes, one of the main motivation to use a soft-processor on FPGA is the possibility to implement fault-tolerant systems with *Commercial Off-the-Shelf* (COTS) components. Depending on the country, the acquisition process of radiation hardened (rad-hard) components is controlled by government agencies. For that matter, the LEON3 [11] processor is used in a few space missions. Moreover, given the Brazil's *National Institute For Space Research* (INPE) interest in migrating from the ERC32 (a discontinued radiation-tolerant SPARC V7 processor developed for space applications) [12] and not having to redesign all the code, the soft-core LEON3 (SPARC V8 based) processor with fault tolerance is a good choice for the rad-hard ERC32.

Additionally, space applications have become increasingly more complex. Hence, some of the computation-intensive tasks can be implemented as special blocks, while the more traditional tasks (e.g., network communication) can be implemented in soft-processor. Nonetheless, the need for more processing capacity is always of interest given the increasing complexity of functions performed in space applications.

This work presents an approach to implement CR on a soft-core processor for FPGA with application-level transparency aimed at space applications. The scheme was implemented in a LEON3 processor embedded in the target FPGA, at the architecture level. The system was simulated using a set of four programs, the results demonstrate that it is possible to keep data consistent after performing checkpoints and recoveries throughout the execution of an application.

The remaining of the paper is organised as follows. Section II describes some of the existing CR techniques. Section III presents an overview of the CR technique. In Section IV the proposed technique using CR is presented. Section V shows the test setup and simulation analysis for our workload. Lastly, section VI concludes the paper and discusses the future work.

II. Related Work

The CR techniques are classified regarding the design abstraction level of the implemented system. CR can be implemented in pure hardware, pure software, or a combination of the two as a hybrid solution. While software-only solutions may be economic from the point of view of overall area cost, the hardware-only approaches trade off area overhead for a lower execution time.

In [13], a solution based on *Dual Modular Redundancy* (DMR) and CR configuration is used in order to find the optimal checkpoint selection. The authors proposed an algorithm that minimises the checkpoint overhead for a given latency constraint.

Some older techniques propose modifications to the cache management algorithm (i.e., cache controller hardware), as in [14, 15]. In this way, the implementation of the CR technique is transparent to the user, while the modified cache management algorithm makes sure the data is consistent between the multiple processors on the system.

ReVive [16] is a general-purpose rollback recovery mechanism for shared-memory multiprocessors. ReVive recovers both transient and permanent faults. ReVive's CR is implemented by special hardware (by modifying the directory controller of the memory) and software. Memory logging (partial separation checkpointing) is implemented in special hardware, while processor context (i.e., register file and other status registers) checkpointing is implemented in software.

SafetyNet [17] is a lightweight checkpoint recovery mechanism considering long-latency fault detection schemes. In comparison to ReVive, SafetyNet uses special checkpoint buffers including checkpoint log buffer (CLB) for memory and register buffer (for registers), instead of reusing the memory as checkpoint buffer.

A hybrid approach is proposed in [18], where the authors aim to optimize the overall system reliability while considering performance/throughput using a three stage process. A design-time application-specific task resiliency analysis is used at first to generate a heterogeneous treatment for core customization, where both spatial and temporal redundancies are used. This heterogeneous error recovery scheme allows a runtime adaptation between different recovery modes considering the individual task constraint.

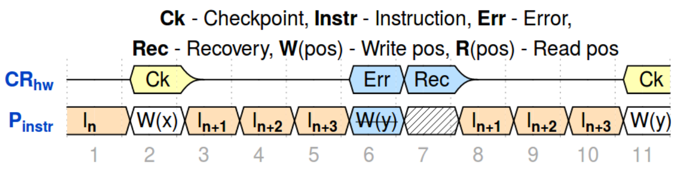
III. Background and Overview

In this paper, the assumed fault model is the *Single Event Effect* (SEE), more precisely its subtype *Single Event Upset* (SEU), since literature shows that SEU is the predominant failure when considering processors [2, 19]. We also consider the hardware as an flash-based FPGA, in our case the Microsemi ProASIC3e FPGA [20]. In this type of FPGA, the configuration memory is not affected by SEUs [21, 22]. Furthermore, this hardware can be migrated to an anti-fuse FPGA, where the configuration memory, after written, cannot be changed, resembling the FPGA to an *Application Specific Integrated Circuit* (ASIC) [23].

The CR technique works by saving checkpoints considered safe during the execution of a processor. Whenever an error is detected, a rollback to the last known safe state is performed, namely recovery. A design decision made is the granularity of the checkpoints, once it introduces overhead in the processor execution. One possible metric that can be used is to assume that after every write operation to the main memory, the state can be saved. Another approach used in [24] is to save a checkpoint before the occurrence of a jump instruction in the execution of the program.

Fig. 1 depicts a hypothetical scenario: after a checkpoint (Ck) is performed at $t = 2$, instructions I_{n+1}, I_{n+2} and I_{n+3} are executed. At time $t = 6$ the error is detected, causing a recovery to occur. After the recovery, the three instructions are executed in the same fashion and the fault is overwritten with the right result.

Since the fault is transient, the SEUs occurs when a wrong value is stored in an element of the circuit, and, if the element is overwritten with the correct value, after the SEUs



Ck - Checkpoint, **Instr** - Instruction, **Err** - Error,
Rec - Recovery, **W**(pos) - Write pos, **R**(pos) - Read pos
CR_hw
Ck
Err
Rec
Ck
P_instr
I_n
W(x)
I_n+1
I_n+2
I_n+3
W(y)
I_n+1
I_n+2
I_n+3
W(y)
1
2
3
4
5
6
7
8
9
10
11

Fig. 1. Checkpoint Recovery Technique Scenario

is identified, the error can be corrected. Therefore, the CR technique, which repeats the operation of a point considered safe, is a reasonable solution.

In order to perform the rollback in the processor, the CR hardware needs to be aware of the error, meaning an error-detection must be implemented. There are several error-detection techniques in the literature. This study does not primarily aim at the detection of a SEU (i.e. error detection), as it can be considered another field of study by itself. Instead, we used fault tolerance techniques, which have fault-detection as their starting point. Three techniques have been used: the classical TMR [25]; a bus-based DMR approach [26]; and a time-redundant execution. The Fig. 2 presents the three architectures used in the experiment.

Fig. 2(a) uses a bus-based DMR to detect errors and inform to the CR module to perform the rollback on both processors; Fig. 2(b) is a classic TMR where it always detects single errors and masks single-faults using a majority voter; Fig. 2(c) employs the time redundant approach that executes twice every slice of code. In this case, the CR module saves the address and data that is going to be written on the main memory on the first attempt. After, rollback is performed and the second address and data generated are compared with the ones saved in the first execution. When there is a match, the memory write operation is performed and a new checkpoint is saved, advancing the code execution to the next slice. If the values do not match, the second execution address and data are also stored by the CR hardware and another rollback is done to have a third execution of the code. This way, the CR hardware can use the result of three executions to perform a simple majority vote (similarly to the TMR) and write to the main memory the correct value. In the case of three different executions, a error signal is raised, similar to the voter error on TMRs approach, bringing the processor to a halt.

IV. Proposed Checkpoint and Recovery Technique

During its normal operation the processor creates checkpoints, which represent consistent states that can be restored. The checkpoints are a copy of the processor current state, more specifically the content of the pipeline registers. Any changes to the registers file since the last consistent checkpoint are saved. The granularity of the checkpoints was designed, in such way that one checkpoint is created every time the processor executes an instruction that perform writes in the main memory. Since the main memory is the reference, instruction and data caches were disabled on the processor configuration. Even thought the absence of cache in the processor degrades overall performance (in terms of execution time), it also introduces another point of failure for SEUs once it is a type of storage element.

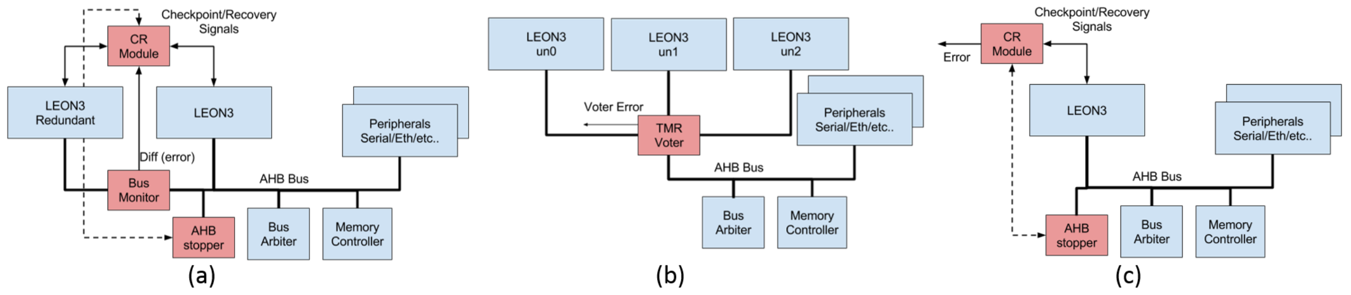


Fig. 2. Different Architectures Used for Error Detection: (a) bus-based DMR, (b) bus-based TMR, and (c) single-processor time-redundant.

In order to implement the CR technique, the LEON3 hardware was modified. The first step was to find all the registers on the pipeline that hold the current state of the processor. In more detail, the *PROC3* unit have VHDL processes, comprising the pipeline, that needed to be saved. Even though the instruction and data caches were disabled, there are *Finite State Machine*s (FSMs) that controlled the communication between the *Integer Unit* (IU) and the *Advanced Microcontroller Bus Architecture* (AMBA) bus that needed to be checkpointed as well. A single checkpoint signal was connected to all modules involved, when the main memory write is detected, it performs the checkpoint by copying all the data to redundant registers.

The register file, likewise, need to be taken into consideration when recovering the processor state. In order to do so, a fourth port was added to the register file to perform a read on the register that is currently being written, this way the old value can be saved in a memory stack. On the recover event, the stack is dumped back into the register file, bringing it back to its safe state (last checkpoint).

Storing redundant data of the processor pipeline and register file (i.e. checkpoint data) make another weak spot for SEUs. At the current stage, the system is being simulated without injecting faults on this information.

The time redundancy is obtained by using the CR mechanism, to run each interval between checkpoints twice (Fig. 2 - (c)). On the first run, the processor saves the information of the memory write instruction, but does not allow it to proceed, bypassing the memory write enable signal. Then, a rollback is performed and the processor executes all instructions from the last checkpoint. When the second run reaches the memory write instruction, the CR mechanism compares address and data to the ones stored from the first run, if they are equal, the main memory is written and the process repeats, otherwise, the fault is detected and a mismatch is signalised.

V. Experimental Analysis

In this section we describe the adopted simulation method, test setup and benchmarks used in our tests to obtain simulation results. We use the fault definition according to Avizienis *et al.* [27].

A. Simulation Method

In order to run our tests, the LEON3 processor was simulated using the Modelsim tool. The fault injection was performed according to the pseudo-algorithm in Fig. 3. The fault injection script reads all LEON3's IU registered signals (memory elements). For each signal a new simulation is run (line 2). In each simulation, a random time is picked (line 3) and ran. After the run time, the current signal value is read (line 4) and a SEU is simulated by inverting one bit inside the signal value (line 5) and applying it to the current signal using a force command (line 6). Note that this force command modifies the signal until it gets overwritten, known as *deposit* on the simulator tool. Finally, the simulation is ran until its end. This means that the program comes to its final state, by raising a stop signal, or an error signal (if detected by the simulation script). In line 10 we make sure we have enough samples to fulfil a confidence interval of 95% and a margin of error less than 5% (since it is a large population: $0.98/\sqrt{n}$, or at least 400 runs).

Fig. 3. Simulation Steps Pseudo-algorithm

```
1   do{
2     foreach(signal current in L3.IU3){
3       run rand();
4       value = read(current);
5       seu(&value);
6       force(current, value);
7       run all;
8       runs++;
9     }
10  }while(runs < CONFIDENCE);
```

The simulation results were classified according to Fig. 4. After fault injection, there are three possible results (outcomes): Correct; Detected; or Failure. A correct result implies in either no error detected, and/or a latent error detection. This means that the fault in that signal, at a given time did not affect the execution. A failure means that the fault causes a failure in the processor without being possible to detect it. Lastly, the detected fault is the result of an error, which can be further classified in three possible situations according to the fault-tolerant technique used: Recovered; Not-recovered; and Recovered incorrectly. A recovered situation is when after detect, the recovery process acts accordingly and the program finishes its execution with the expected result. A not-recovered error happens when the recovery process fails to finish the program, either without the expected result or a time-out. The last case is when the recovery process is performed and the program reaches its final state with an incorrect result. This can happen when the error occurs on the variable that controls a loop, for example.

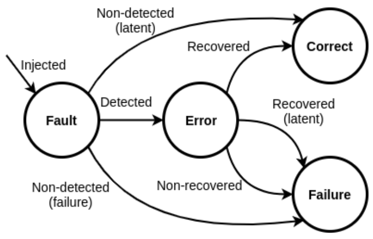


Fig. 4. Fault States

B. Experimental Setup

For each architecture of our tests a set of four programs were used to stress the processor architecture as follows:

1) Basic: simple arithmetic operation that is executed 50 times and checked against the correct value.
2) Bubble sort: classic bubble sort algorithm on a 10 element vector.
3) NMEA: calculate the checksum (bitwise XOR) of ASCII codes on a message string.
4) Hamming: calculate an hamming encoded message using a matrices.

It is important to note that we did not use a more classic test program (such as *dhry*, *stanford*, or *whetstone*) since the simulation time was prohibitive, *e.g.* over a day on a high-end computer for a single execution.

C. Detection and Recovery Analysis

Fig. 5 presents the detection analysis for the three architectures used in the experiment with the inclusion of the LEON3 original (unmodified) configuration. Note that for the original configuration there is no detection available, therefore only the Correct/Failure results are presented.

For the architectures of the TMR an the DMR the detection rate were almost the same, in the order of 79%, with no failures detected, which means that either the fault is latent, or is detected. The failure rate of the original is slightly lower than the detected figures in the TMR and DMR approaches, this is due to a detected error not always becoming a failure.

Interestingly the time redundant approach shows the higher percentage of corrected results (in the order of 95%). This is due to the re-execution of the code slice, meaning that since the injected fault can be overwritten it may not manifest itself during the program execution.

Fig. 6 shows an analysis for the recovery process on the detected errors for the time redundant and DMR approaches. The TMR is not shown since it has 100% correction for single faults.

D. Execution Overhead Analysis

Although the CR technique presents a competitive recovery capability, it introduces time overhead on the program execution. Whenever a recovery is done, the execution needs to be halted for at least one clock cycle, allowing the recovery of the IU pipeline registers and an additional clock cycle for each register in the register file used since the last safe checkpoint.

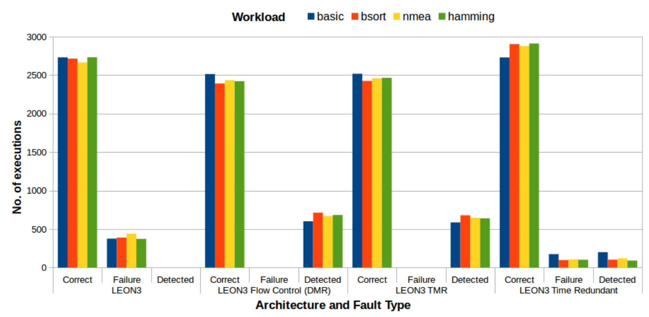


Fig. 5. Detection analysis

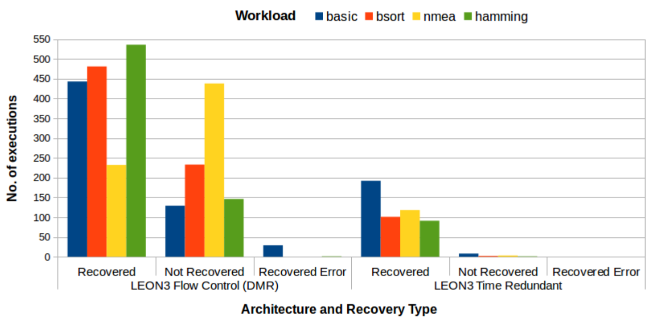


Fig. 6. Recovery analysis

TABLE I. EXECUTION TIME OVERHEAD AGAINST BASELINE

	LEON3 Flow Control		LEON3 Time Redundant	
	Correct	**Recovered**	**Correct**	**Recovered**
basic	0%	2.05%	52.70%	53.04%
bsort	0%	0.23%	94.09%	94.18%
nmea	0%	0.85%	81.86%	82.03%
hamming	0%	0.85%	84.61%	84.83%
Average	0%	0.99%	78.31%	78.52%

Each implementation of the LEON3 have different time overhead on the execution. The LEON3 Flow Control does not add time to perform the checkpoints (since the checkpoint procedure is done in parallel), therefore no overhead is noticed when there are no errors detected on the execution. Once an error is detected, the recovery procedure takes a few clock cycles to occur, hence the average value of 0.99% of time increase against the baseline execution.

The cost of executing twice each slice of code out-stands on the LEON3 Time Redundant approach. On the average it adds 78.31% for a correct execution and 78.52% when the error is detected. It is important to note that the time redundancy is only enabled after the processor reaches the user program area, this is due to the memory that stores the initialization code is a ROM, hence no writes on the main memory are performed.

E. FPGA Area Overhead Analysis

Although we did not performed test on the FPGA hardware, it would be interesting to compare how the different architectures influence on the area occupied. In order to do so, we ran the synthesis tool on a Xilinx FPGA (Spartan-3 1500, model xc3s1500-4-fg456). The reason for the use of a different FPGA from the aforementioned (Microsemi

TABLE II. AREA OVERHEAD COMPARISON FOR A XILINX SPARTAN-3 1500 FPGA

Resource Type	Available	Baseline	Time Redundant	Increase (%)	DMR	Increase (%)	TMR	Increase (%)
Slices	13312	6483	8208	26.61%	9873	52.29%	13702	111.35%
Slices FF	26624	3208	5807	81.02%	6145	91.55%	6512	102.99%
4-input LUT	26624	12017	12598	4.83%	18424	53.32%	25836	115.00%
BRAM	32	6	7	16.67%	10	66.67%	14	133.33%
MULT18x18	32	4	4	0%	8	100.00%	12	200.00%

ProASIC3e) is that the LEON3's GRLIB does not provide models for simulation of the main memory. Nonetheless, all modifications were made inside the LEON3 core architecture, hence no difference should be found when synthesized to the Microsemi device.

Table II shows the total of device resources used for the different architectures compared to the baseline (no modifications) and also the available resources on the FPGA. For all resources presented, the lower overhead obtained can be noticed on the Time-Redundant variant, indeed it was expected, since no processor replication was made. Respectively, the DMR and TMR have higher occupation rates when compared with the Time-Redundant approach.

The Time-Redundant version have a significant increase in Slices due to logic implementation, but it is the most abundant resource on the FPGAs. Nonetheless, when coupled with the results of time overhead, this technique presents a competitive approach when compared to the DMR and TMR versions. More over, the spare logic on the FPGA could be used to implement other functions or improve even further the technique by protecting more elements on the processor.

VI. CONCLUSION

This paper presented and approach to implement checkpoint recovery on a soft-core processor, aiming application-level transparency. The technique was implemented on different architectures and a fault-injection campaign were run to validate against more traditional fault-tolerant approaches.

The implemented approach of the CR technique shows that it is possible to keep the processor execution consistent. The fault tolerance was significantly improved on both architectures using the CR.

As future work, this technique needs to be fully synthesized and implemented on the target FPGA. With this implementation we aim to run heavy-ion experimentation in order to evaluate the technique robustness against SEEs.

ACKNOWLEDGEMENT

This work has been partly funded by the **Brazilian National Council for Scientific and Technological Development** (CNPq) and **Instituto Federal do Rio Grande do Sul** (IFRS).

REFERENCES

[1] M. Gordon, P. Goldhagen, K. Rodbell, T. Zabel, H. Tang, J. Clem, and P. Bailey, "Measurement of the flux and energy spectrum of cosmic-ray induced neutrons on the ground," *IEEE Trans. Nucl. Sci.*, vol. 51, no. 6, pp. 3427–3434, 12 2004. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1369506

[2] M. Reorda, M. Violante, C. Meinhardt, and R. Reis, "A low-cost SEE mitigation solution for soft-processors embedded in Systems on Programmable Chips," *2009 Design, Automation & Test in Europe Conference & Exhibition*, pp. 352–357, 4 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1874620.1874704

[3] R. E. Lyons and W. Vanderkulk, "The Use of Triple-Modular Redundancy to Improve Computer Reliability," *IBM Journal of Research and Development*, vol. 6, no. 2, pp. 200–209, 4 1962. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5392355

[4] S. Punnekkat and A. Burns, "Analysis of checkpointing for schedulability of real-time systems," in *Proceedings Fourth International Workshop on Real-Time Computing Systems and Applications*. IEEE Comput. Soc, 1997, pp. 198–205. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=629219

[5] C. A. Kletzing, W. S. Kurth, M. Acuna, R. J. MacDowall, R. B. Torbert, T. Averkamp, D. Bodet, S. R. Bounds, M. Chutter, J. Connerney, D. Crawford, J. S. Dolan, R. Dvorsky, G. B. Hospodarsky, J. Howard, V. Jordanova, R. A. Johnson, D. L. Kirchner, B. Mokrzycki, G. Needell, J. Odom, D. Mark, R. Pfaff, J. R. Phillips, C. W. Piker, S. L. Remington, D. Rowland, O. Santolik, R. Schnurr, D. Sheppard, C. W. Smith, R. M. Thorne, and J. Tyler, "The Electric and Magnetic Field Instrument Suite and Integrated Science (EMFISIS) on RBSP," *Space Science Reviews*, vol. 179, no. 1-4, pp. 127–181, 6 2013. [Online]. Available: http://link.springer.com/10.1007/s11214-013-9993-6

[6] R. Glein, "BRAM Radiation Sensor for a Self-Adaptative SEU Mitigation," in *SpacE FPGA Users Workshop*, 2014.

[7] D. S. Wilson, "Cubesat Flight Software Development," in *2011 Workshop on Spacecraft Flight Software (FSW11)*, Baltimore, 2011.

[8] D. Guzmán, D. E. Rowland, P. Uribe, and T. Nieves, "A Low Power Processors for Cubesat Missions," in *8th Annual Cubesat Developers Workshop 2011*, 2011.

[9] C. Bernardeschi, L. Cassano, and A. Domenici, "SRAM-Based FPGA Systems for Safety-Critical Applications: A Survey on Design Standards and Proposed Methodologies," *Journal of Computer Science and Technology*, vol. 30, no. 2, pp. 373–390, 3 2015. [Online]. Available: http://link.springer.com/10.1007/s11390-015-1530-5

[10] D. Sabena, L. Sterpone, M. Scholzel, T. Koal, H. T. Vierhaus, S. Wong, R. Glein, F. Rittner, C. Stender, M. Porrmann, and J. Hagemeyer, "Reconfigurable high performance architectures: How much are they ready for safety-critical applications?" in *2014 19th IEEE European Test Symposium (ETS)*. IEEE, 5 2014, pp. 1–8. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6847820

[11] Aeroflex Gaisler, "LEON3 Processor," 2015. [Online]. Available: http://www.gaisler.com/index.php/products/processors/leon3

[12] ESA, "European Space Agency," 2014. [Online]. Available: www.esa.int/

[13] S.-h. Kang, H.-w. Park, S. Kim, H. Oh, and S. Ha, "Optimal Checkpoint Selection with Dual-Modular Redundancy Hardening," *IEEE Transactions on Computers*, vol. PP, no. 99, pp. 1–1, 2014. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6880324

[14] R. Ahmed, R. Frazier, and P. Marinos, "Cache-aided rollback error recovery (CARER) algorithm for shared-memory multiprocessor systems," in *Digest of Papers. Fault-Tolerant Computing: 20th International Symposium*. IEEE Comput. Soc. Press, 1990, pp. 82–88. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=89338

[15] K.-L. Wu, W. Fuchs, and J. Patel, "Error recovery in shared memory multiprocessors using private caches," *IEEE Transactions on Parallel and Distributed Systems*, vol. 1, no. 2, pp. 231–240, 4 1990. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=80134

[16] M. Prvulovic and J. Torrellas, "ReVive: cost-effective architectural support for rollback recovery in shared-memory multiprocessors," in *Proceedings 29th Annual International Symposium on Computer Architecture*. IEEE Comput. Soc, 2002, pp. 111–122. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1003567

[17] D. Sorin, M. Martin, M. Hill, and D. Wood, “SafetyNet: improving the availability of shared memory multiprocessors with global checkpoint/recovery,” in *Proceedings 29th Annual International Symposium on Computer Architecture*. IEEE Comput. Soc, 2002, pp. 123–134. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1003568

[18] T. Li, M. Shafique, S. Rehman, J. A. Ambrose, J. Henkel, and S. Parameswaran, “DHASER: dynamic heterogeneous adaptation for soft-error resiliency in ASIP-based multi-core systems,” in *ICCAD ’13 Proceedings of the International Conference on Computer-Aided Design*. IEEE Press, 11 2013, pp. 646–653. [Online]. Available: http://dl.acm.org/citation.cfm?id=2561828.2561955

[19] L. Lesage, B. Mejias, and M. Lobelle, “A software based approach to eliminate all SEU effects from mission critical programs,” in *2011 12th European Conference on Radiation and Its Effects on Components and Systems*. IEEE, 9 2011, pp. 467–472. [Online]. Available: http://ieeexplore.ieee.org/document/6131353/

[20] Microsemi Inc., “ProASIC3 FPGA,” 2017. [Online]. Available: https://www.microsemi.com/products/fpga-soc/fpga/proasic3-overview

[21] P. R. C. Villa, R. C. Goerl, F. Vargas, L. B. Poehls, N. H. Medina, N. Added, V. A. P. de Aguiar, E. L. A. Macchione, F. Aguirre, M. A. G. da Silveira, and E. A. Bezerra, “Analysis of single-event upsets in a Microsemi ProAsic3E FPGA,” in *2017 18th IEEE Latin American Test Symposium (LATS)*. IEEE, 3 2017, pp. 1–4. [Online]. Available: http://ieeexplore.ieee.org/document/7906772/

[22] P. Villa, E. Bezerra, R. Goerl, L. Poehls, F. Vargas, N. Medina, N. Added, V. De Aguiar, E. MacChione, F. Aguirre, and M. Da Silveira, “Analysis of COTS FPGA SEU-sensitivity to combined effects of conducted-EMI and TID,” in *Proceedings of the 2017 11th International Workshop on the Electromagnetic Compatibility of Integrated Circuits, EMCCompo 2017*, 2017.

[23] J. McCollum, “ASIC versus antifuse FPGA reliability,” in *2009 IEEE Aerospace conference*. IEEE, 3 2009, pp. 1–11. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4839526

[24] R. Ragel and S. Parameswaran, “Reli: Hardware/software Checkpoint and Recovery scheme for embedded processors,” in *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 3 2012, pp. 875–880. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6176621

[25] V. M. G. Martins, P. R. C. Villa, H. C. C. Neto, and E. A. Bezerra, “A TMR Strategy with Enhanced Dependability Features Based on a Partial Reconfiguration Flow,” in *2015 IEEE Computer Society Annual Symposium on VLSI*. IEEE, 7 2015, pp. 161–166. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7309556

[26] F. Ferlini, F. A. da Silva, E. A. Bezerra, and D. V. Lettnin, “Non-intrusive fault tolerance in soft processors through circuit duplication,” in *2012 13th Latin American Test Workshop (LATW)*. IEEE, 4 2012, pp. 1–6. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6261264

[27] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, “Basic concepts and taxonomy of dependable and secure computing,” *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, 1 2004. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1335465