A Fault-Tolerant Time-Predictable Processor

Christos Gkiokas, Martin Schoeberl

Department of Applied Mathematics and Computer Science
Technical University of Denmark
Email: s182348@student.dtu.dk, masca@dtu.dk

***Abstract*—Advancements in commercial space applications have increased interest in deploying FPGA devices with soft processors on satellite systems. Processors are a vital component in all satellite payloads used for control functions and on-board data processing. However, a high integration level within an FPGA causes high sensitivity to ionising radiation-induced errors, requiring mitigation for single event effects to ensure reliable operation. In this paper, a triple-core lock-step processor for radiation-induced soft-errors mitigation is presented that aims to reduce the probability of a processor failure due to soft-errors. Aerospace applications are also defined by strict real-time requirements for processors, thus the design is implemented using Patmos, a time-predictable processor of the T-CREST multicore research platform.**

***Index Terms*—Fault-tolerance, triple-modular redundancy, lock-step, single event upsets, time-predictable architecture.**

I. Introduction

Every modern aerospace system is depending on processors to perform vital control and data handling operations. Currently, this task depends on purpose-built radiation hardened processors that are priced extremely high and their designs are often decades behind what is commercially available. The current trend in commercial and industrial applications is to integrate large systems into a single device known as System-on-Chips (SoC). These systems are particularly suited to be implemented using FPGAs (field programmable gate array) and are much more cost-effective than a custom chip in low production volumes required in aerospace systems. A soft processor that is implemented in such a device, is a very compelling choice that allows for processor specific customisation and custom reliability features. Additionally, FPGA devices provide a significant amount of high bandwidth I/Os and are capable of having re-configurable logic, providing adaptability and flexibility while in the field [15].

Aerospace systems are not only fault-tolerant systems, but they also have strict real-time requirements. The worst-case execution time of a software task must be bound and known, guaranteeing that the response time of critical functions is delivered in time. With this in mind, this paper presents a triple-core lock-step processor by extending the time-predictable processor Patmos. Patmos is an open-source microprocessor specifically designed to support worst-case execution time analysis and offer compatibility to the predictability requirements of hard real-time systems [13]. Patmos is part of the multicore platform T-CREST [12].

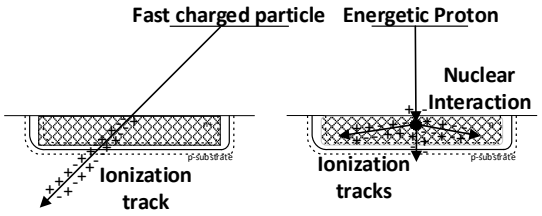

Fig. 1. Cosmic ray energy deposition on silicon gate [10]

To provide complete single event effect (SEE) mitigation on an FPGA SoC, the lockstep approach is not adequate on its own. Mitigation techniques to protect the main memory of the system and the FPGA configuration memory must be applied as well. Typically these include memory scrubbing [9] and partial or complete FPGA reconfiguration schemes [5], [2]. This paper is focused on the design of the triple module redundancy (TMR) lockstep architecture for the processor, thus it does not examine single event effect protection outside of the processor core.

The remainder of this paper is organised as follows: Section II presents the background on SEEs and common system redundancy techniques. Section III reviews works related to this paper. Section IV presents the design and implementation of the proposed system. Section V presents the results. Section VI concludes the paper.

II. Background

FPGA devices are even more sensitive to ionising radiation effects as, to achieve the aforementioned customisation levels, as their configuration depends on memory elements like SRAM. These effects are called single event effects, a subcategory of these effects is single event upsets (SEU) being the most relevant in digital circuits. SEUs are caused when charged particles are hitting the silicon substrate inside integrated circuits leaving an ionisation trail, or energetic particles like neutrons collide with the substrate and via nuclear interaction releasing a shower of charged particles leaving individual trails. The electron-hole pairs produced by this interaction corrupts stored information in memory elements or flip-flops. These are called soft-errors as they are not damaging to the device and can be corrected by resetting the affected value [19], [8]. This interaction is depicted in

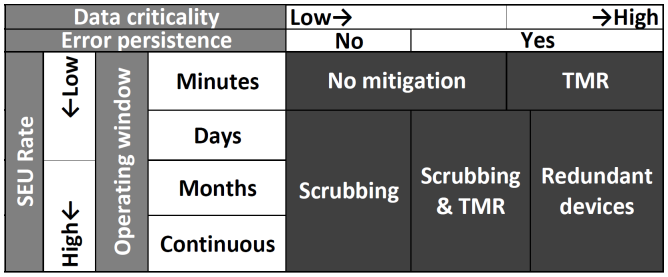

Fig. 2. Mitigation techniques for memories depending on SEU rate and mission criticality [16]

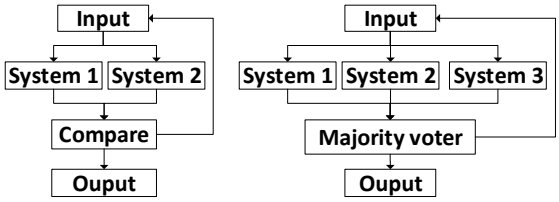

Fig. 3. DMR vs TMR, in DMR the results are compared against while in TMR the result is decided by a majority voter circuit

Fig. 1. Specifically, SEUs can corrupt the functionality of an FPGA by altering the internal state of individual flip-flops and memory blocks, or affect the state of custom logic elements inside the FPGA by corrupting look-up tables, phased locked loops, digital signal processing slices, etc.

Over the years, several techniques have been proposed and implemented to mitigate such effects, such as employing error correcting codes or memory scrubbing. In 2 common memory soft-error mitigation schemes are presented in relation to SEU rate, operational time requirements and data criticallity. This paper is focused in the use of redundant devices, specifically TMR. The two most common redundant systems schemes used are double module redundancy (DMR), TMR and in extreme cases, quadruple module redundancy may be used. Fig. 3 shows typical DMR and TMR architectures, in both cases the inputs is identical to all systems and the output is either compared for DMR or voted upon in TMR. Although redundancy always comes with the price of extra resource overhead, greater granularity of redundancy is considered more reliable. The most popular technique in aerospace systems is the use of TMR where three identical systems, are operating concurrently and their results are compared by a majority voter [11].

The lock-step architecture is a case of DMR or TMR specifically for processors. “Lock-step” in military terms is used to describe synchronised marching, similarly here it is used to describe two or three processors running in parallel, using identical inputs and clocks. All processors must start at the same time and their states and data must be equal every clock cycle. Errors are detected by the output comparison/majority voter unit, if the two processors have different results or if the majority voter fails to reach consensus respectively. A major difference between TMR and DMR lock-step systems is that in TMR if one of the processors is faulty there is knowledge of the faulty processor, due to the triplicated process and the execution can proceed uninterrupted. In contrast, DMR with only two results, it is impossible to decide which one is fault-free. In the event of error detection, the system halts execution and typically performs a rollback operation, meaning that the processors are reset to the last known fault-free state and resume execution from there. This is achieved by keeping a processor architectural state context periodically and store it for the event of a rollback. It is evident however that this technique introduces considerable overhead and performance penalties by tripling/doubling the resources needed without any increase in processing power. So in practice, lock-step systems are designed with the balance between resource overhead and reliability in mind. In this paper, a system is presented that attempts to implement TMR lock-step for a time-predictable processor by employing three Patmos cores in triple-core lock-step. The cores execute in parallel and the results are checked with the use of a majority voter.

III. Related Work

There have been attempts to introduce lock-step processors in recent years, using either two-core or three-core processor setups. In [7], a TMR based design is presented as an improvement to an existing dual-core lock-step. The system uses a shared instruction cache that is read by the triplicated cores and their outputs are checked by a majority voter. It also employs a separate error detection mechanism that classifies detected errors to correctable or non-correctable and triggers re-synchronisation of the architectural state (register file, program counter, state registers) to a previously stored stable state, when a non-correctable error is detected. Additionally, this architecture is designed to also be used in silicon and not only as a soft-core for FPGAs.

Another approach is presented in [3], where a DMR architecture is presented. The processors use shared data and program memories but in this design, error correction codes are used after each step of the pipeline, while the outputs are checked with a DMR comperator. Additionally, data are checked for lock-step errors in each stage of the pipeline. If an error is detected then a “context” reload is issued which similarly to [7] restores the cores to a previously saved stable state.

Similarly, in [17] a TMR based solution is presented for a soft processor used in satellite systems. In this design, the processor is triplicated and majority voters are placed between the cores that do not share any architectural elements as seen in the previous setups. Also, the global clock is triplicated into independent clock buffers, leading to each soft processor using a unique clock. Furthermore, this design does not incorporate any rollback features and handles disagreements externally, by scrubbing and completely rewriting the configuration memory of the FPGA.

In [6], a TMR based scheme is presented using triplicated soft processors that are compared using a majority voter at the

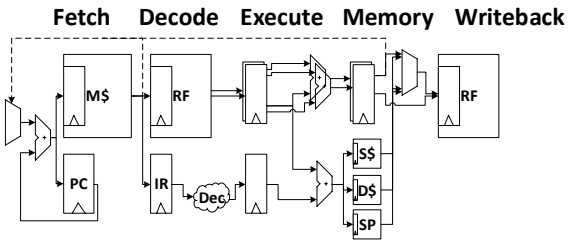

Fig. 4. Overview of Patmos processor pipeline [14]

output and input from the main memory. Again there are no shared components between the cores and the error checking is implemented either at the input or output of the processors. As in [17], there is no rollback technique used for recovery, the authors propose partial reconfiguration of the FPGA in case of disagreement.

Furthermore, in [4] another DMR approach is presented using FPGA reconfiguration instead of rollback to handle disagreement. In contrast to all the previous techniques, this implementation uses the two cores in a master-slave configuration where the input is duplicated, but the output is only propagated to the main memory from the master. The outputs are validated outside of the data path by a comperator that is only raising an error signal to trigger the reconfiguration process.

Finally, the proposed design is also TMR based. In contrast to the works presented in this section, the TMR is formed by triplicating pipeline elements, extending the existing soft-core processor. This is achieved by placing the majority voter between the execute and memory stages of the processor pipeline. The processors use a common clock as well as a common instruction cache and memory stage. In case of full disagreement, the processor will try to recover by resetting the program counter to the address of the instruction that caused the failure at execute stage and by flushing the decode and execute stages.

IV. Design and Implementation

The design presented in this paper implements a triple-core lock-step system using three synchronised cores checked by a majority voter circuit. The processor cores used in this design is a 32-bit, dual-issue, statically scheduled, RISC-style microprocessor optimised for time-predictable execution of real-time applications [14]. It consists of five pipeline stages: instruction fetch, decode, execute, memory, and register write back. The architectural schematic of this processors pipeline is depicted in Fig. 4. Furthermore, the processor is capable of multicore operation with the use of an external time-division multiplexed (TDM) arbiter, that enforce static schedule access to the main memory.

The TMR in this design is created by triplicating the pipeline and inserting a majority voter circuit between the execute and memory stages of the pipeline. In Fig.5 a block diagram of the TMR scheme is depicted, the cores share the

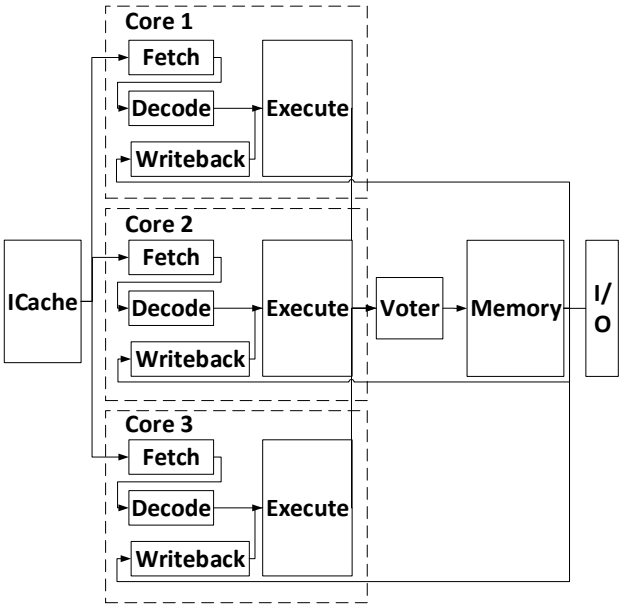

Fig. 5. The triplicated pipeline design

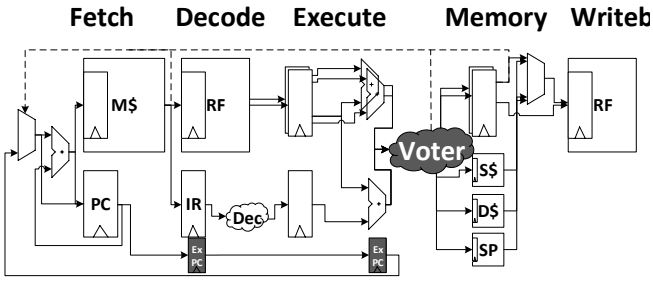

Fig. 6. Connection of the voter component into the pipeline

instruction cache as well as the memory and input/output component. The results of the execute stage are passed through the majority voter and then propagated to the shared memory stage, thus guaranteeing that no erroneous results can contaminate the shared registers. The results are kept in lockstep by synchronising a data valid signal for each core that allows the data to propagate only if all the cores are ready and agreeing. In addition, a shared instruction cache is used to ensure that inputs of the cores are truly in lockstep. Finally, the I/O component is also shared to enable the design to function with all the external components available for the Patmos processor, as the interface remains the same as if it was still single core. Most importantly, this allows the TDM arbiter to issue a single schedule slot per lockstep core, enabling multicore operation with lockstep cores.

Figure 6 presents the placement of the voter component. The voter is connected inside the pipeline, right after the execution stage and collects the outputs as the execution progresses. If the results of the cores are error-free then the data flow freely

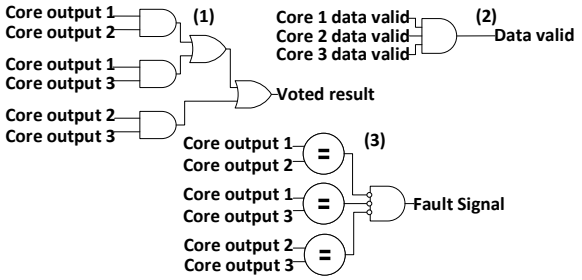

Fig. 7. The voter component: (1): majority voter. (2): data valid synchronisation. (3): three core disagreement detection.

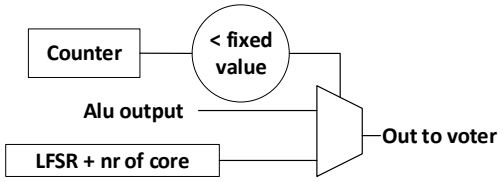

Fig. 8. Pseudo-random test vector injection circuit

into the memory stage of the pipeline. This design has the drawback that it places the voter and error detection circuitry inside the critical path of the processor. The reason behind this design decision is to stop any erroneous data to propagate from the execution stage to any memory elements as there is no stable context backup to revert to. There can be two cases of core result disagreement, if one core presents a non-agreeing result and if all cores are in disagreement. In the first case, the majority circuit inside the voter component propagates the correct by majority result to the memory stage and execution resumes normally. In the second case, a fault signal is raised by the voter component that resets the program counter in the fetch stage to the program counter value that was executing at the time disagreement occurred. Furthermore, the fault signal triggers a flush signal that flushes the decode and execute stages, so that the instruction can be executed again in a clean pipeline.

A. Voter

The voter component consists of three purely combinational circuits that perform the per bit majority voting and error detection for the 32-bit outputs of the execution stage, the synchronisation of the individual execution stages to the memory stage, and the three core disagreement detection circuit, as shown in Fig. 7.

The majority voter has three cases of operation: error-free, single-core error, and non-correctable error. In the error-free case, meaning all three core outputs are equal, the voter forwards the results into the memory stage and continues normal execution. In the single-core error case the result that won the majority vote and is consequently considered error-free, is propagated to the memory stage to continue normal execution. Finally in the non-correctable error case, meaning all three core outputs disagree with each other, the disagreement detection circuit takes over recovery actions, see Fig. 7 (1). The disagreement detection circuit raises the fault signal when all three cores fail a comperator check, see Fig. 7 (3). The fault signal issues the reset of the program counter to the instruction address that failed in the execute stage failed and raises the flush signal.

The result is normally written to the memory stage and the write action is conducted by three signals, the data bus, the address bus and a data valid bit. The voter has to synchronise three data valid signals into one that is returned as the memory input, therefore synchronising memory writes across all cores. As the cores execute the same operations and the valid signal was simply reduced using and gates, see Fig. 7 (2).

B. Testing

In order to test that the majority voter is functional, two ways to inject testing errors were used. Firstly, a 32-bit pseudo-random number generator was designed by adding the output of a linear feedback shift register to the core index number, see Fig. 8. The circuit is controlled via a counter that is compared to a fixed value, which represents the number of clock cycles after which the error vector will take effect. After injection, the counter is reset and operation resumes normally until the counter reaches the set value. To inject the test vectors, a multiplexer is added to the outputs of the cores and is controlled by the comperator that is checking the counter and the fixed value. This test however, is used to test the behaviour of the voter in the three-core disagreement case.

An alternative method that we used for testing was to inject each core with manually corrupted compiled code (binary) files, this method offered a more realistic way of producing errors, simulating bit-flips in the FPGA configuration SRAM. Moreover, it offers more control than random number generation, as the binary file can be edited with wrong but valid instructions, that can be issued at any time of program execution and can test any core failure combination.

V. Evaluation

The presented design was implemented successfully in an Altera Cyclone IV FPGA device. The experimental setup uses the triple-core lock-step cores presented, connected to LEDs and UART modules that were used to execute a simple program written in C. The program simply flashes the LEDs with a frequency of 1 Hz and outputs to the UART the state of the LEDs and was executed successfully with a clock frequency of 50 MHz. Test vector injection was initially set to inject error vectors on all cores after five million clock cycles and verify that the circuit functions as designed.

The modified binary file technique was used to produce the following waveform diagrams. The binary files were manipulated to issue a different operation instruction to the period calculation of the LED flash.

In Fig. 9 shows a single-core error output case. At the first marker on the waveform, a disagreeing output is observed

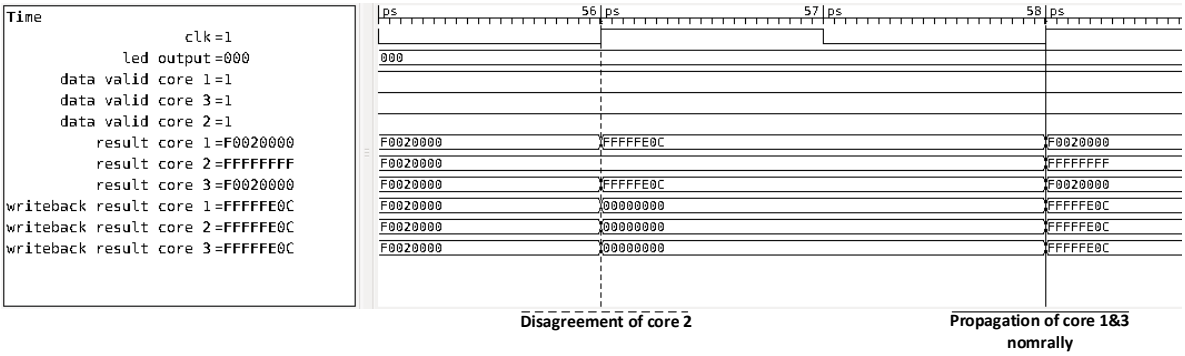

Fig. 9. One core erroneous output

TABLE I
IMPLEMENTATION METRICS COMPARISON BETWEEN A SINGLE PATMOS PROCESSOR, A MULTICORE, AND THE LOCK-STEP IMPLEMENTATION

	Single	Multicore	TMR Processor
Logic elements	9844 (9%)	28467 (25%)	16717 (15%)
Registers	4725	13273	6728
Slack 85°C (ns)	0.109	-0.03	-7.277
Slack 0°C (ns)	1.016	0.792	-5.805
Fmax 85°C (MHz)	80.7	79.81	50.56
Fmax 0°C (MHz)	87.08	86.31	54.63

TABLE II
COMPARISON WITH TWO OF THE PRESENTED SOFT-CORE LOCKSTEP DESIGNS

	Patmos	Wirthlin et al. [17]	Ichinomiya et al. [6]
Logic elements	16717	3824	1376
Fmax (MHz)	50.56	73.7	64.918

in core 2, the data valid signals are raised so the cores are synchronised and the results propagate freely. At the next marker, the majority voter propagates through the pipeline the majority result without stopping.

Fig. 10 presents a three core disagreement case, where all cores provide conflicting results. The instruction causing the malfunction as seen at the first marker is `1A`. In the following clock cycle, instruction `1A` reaches the execution stage where the cores output three different results. We see the fault signal of the voter raising and issues a flush of the pipeline along with the signal to reset the program counter to instruction `1A`. The program counter reset is seen in the next clock cycle market where the instruction is again `1A`, it will take two clock cycles to reach the execution stage at the next marker.

Table I shows a comparison of logic elements and the clock frequency between a single core Patmos processor, a three-core multicore setup, and the three-core lock-step system. The area and registers reported are 10 % higher than the multicore setup and 9 % from the base processor. This is due to the triplication of the pipeline, the shared memory and instruction cache and the absence of the TDM arbiter that is needed for multicore operation. However, it is clear that the voter unit is located in the critical path of the pipeline as it is severely affecting the timing estimates for the lock-step version, with a maximum of $-7.277ns$ slack in comparison to only $-0.03ns$ and $0.109ns$ at 85°C for the other two.

Table II illustrates a comparison of logic elements and max clock estimation between the proposed design and two of the processors presented in Section III. The other two designs were implemented in Xilinx devices and provided area metrics using Xilinx slices, but this design uses Altera logic elements. However, from [18] and [1] we can see that a slice is comprised of four LUTs while the logic element contains only one. Therefore the values presented in Table II have been adjusted to reflect the logic element equivalent.

Finally, the post routing power estimation made by the synthesis tool estimated that the total power dissipation is at 401.07 mW. The voter component contributes 3.82 mW or 0.95 % of the total power consumption

A. Source Access

The source code of this implementation is freely available at the repository: https://github.com/cgkiokas/patmos/tree/master/hardware/src/main/scala/patmos

VI. CONCLUSION

We proposed a triple-core lock-step processor for soft-error mitigation. The proposed processor was implemented and evaluated with success in the Cyclone IV FPGA. The system was running a test program at a frequency of 50 MHz and it recovered successfully from error test vector injection at a rate of five million clock cycles. However, it was also evident that the maximum clock frequency was affected, as shown by the relatively large negative slack results. Consequently, as future work, the introduction of a registered voting design could be tested to reduce the impact of the majority voter circuit on the processor critical path. Additionally, while the design is capable of operating as a multicore processor using lockstep cores, further investigation can be useful for

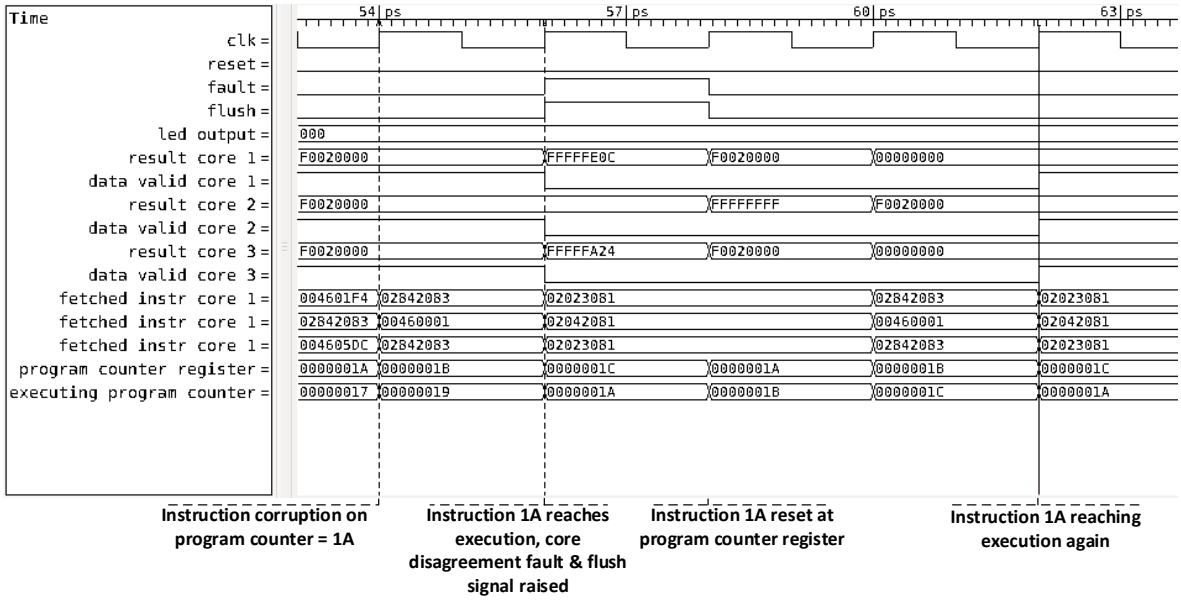

Fig. 10. Three core disagreement

additional mitigation on the TDM arbiter that is used on multicore Patmos systems. Finally to be complete as a system, an error-correcting scheme could be implemented to the main memory by employing error-correcting codes or TMR voting on read/write operation with memory scrubbing.

Acknowledgement

The work presented in this paper was partially funded by the Danish Council for Independent Research | Technology and Production Sciences under the project PREDICT[1] (no. 4184-00127A).

REFERENCES

[1] Altera. Logic elements and logic array blocks in cyclone iv devices. Nov. 2009.

[2] Melanie D Berg and Kenneth A LaBel. Nasa electronic parts and packaging (nepp) field programmable gate array (fpga) single event effects (see) test guideline update. 2018.

[3] Julen Gomez-Cornejo, Aitzol Zuloaga, Uli Kretzschmar, Unai Bidarte, and Jaime Jimenez. Fast context reloading lockstep approach for seus mitigation in a fpga soft core processor. In *IECON 2013-39th Annual Conference of the IEEE Industrial Electronics Society*, pages 2261–2266. IEEE, 2013.

[4] Ahmed Hanafi, Mohammed Karim, and Abdelilah El Hammami. Dual-lockstep microblaze-based embedded system for error detection and recovery with reconfiguration technique. In *2015 Third World Conference on Complex Systems (WCCS)*, pages 1–6. IEEE, 2015.

[5] Jonathan Heiner, Benjamin Sellers, Michael Wirthlin, and Jeff Kalb. Fpga partial reconfiguration via configuration scrubbing. In *2009 International Conference on Field Programmable Logic and Applications*, pages 99–104. IEEE, 2009.

[6] Yoshihiro Ichinomiya, Shiro Tanoue, Motoki Amagasaki, Masahiro Iida, Morihiro Kuga, and Toshinori Sueyoshi. Improving the robustness of a softcore processor against seus by using tmr and partial reconfiguration. In *2010 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*, pages 47–54. IEEE, 2010.

[1] http://predict.compute.dtu.dk/

[7] Xabier Iturbe, Balaji Venu, Emre Özer, and Shidhartha Das. A triple core lock-step (tcls) arm® cortex®-r5 processor for safety-critical and ultra-reliable applications. 07 2016.

[8] Robert Katz, K LaBel, JJ Wang, B Cronquist, R Koga, S Penzin, and G Swift. Radiation effects on current field programmable technologies. *IEEE Transactions on Nuclear Science*, 44(6):1945–1956, 1997.

[9] Eleftherios Kyriakakis, Kalle Ngo, and Johnny Öberg. Mitigating single-event upsets in cots sdram using an edac sdram controller. In *2017 IEEE Nordic Circuits and Systems Conference (NORCAS): NORCHIP and International Symposium of System-on-Chip (SoC)*, pages 1–6. IEEE, 2017.

[10] M Lauriente, A Vampola, R Koga, and R Hosken. Spacecraft anomalies due to the radiation environment. In *36th AIAA Aerospace Sciences Meeting and Exhibit*, page 1048, 1998.

[11] Mahtab Niknahad, Oliver Sander, and Juergen Becker. Fine grain fault tolerance—a key to high reliability for fpgas in space. In *2012 IEEE Aerospace Conference*, pages 1–10. IEEE, 2012.

[12] Martin Schoeberl, Sahar Abbaspour, Benny Akesson, Neil Audsley, Raffaele Capasso, Jamie Garside, Kees Goossens, Sven Goossens, Scott Hansen, Reinhold Heckmann, Stefan Hepp, Benedikt Huber, Alexander Jordan, Evangelia Kasapaki, Jens Knoop, Yonghui Li, Daniel Prokesch, Wolfgang Puffitsch, Peter Puschner, André Rocha, Cláudio Silva, Jens Sparsø, and Alessandro Tocchi. T-CREST: Time-predictable multi-core architecture for embedded systems. *Journal of Systems Architecture*, 61(9):449–471, 2015.

[13] Martin Schoeberl, Wolfgang Puffitsch, Stefan Hepp, Benedikt Huber, and Daniel Prokesch. Patmos: A time-predictable microprocessor. *Real-Time Systems*, 54(2):389–423, Apr 2018.

[14] Martin Schoeberl, Pascal Schleuniger, Wolfgang Puffitsch, Florian Brandner, Christian W Probst, Sven Karlsson, and Tommy Thorn. Towards a time-predictable dual-issue microprocessor: The patmos approach. In *Workshop on Bringing Theory to Practice: Predictability and Performance in Embedded Systems*. OASICS, 2011.

[15] Jason G Tong, Ian DL Anderson, and Mohammed AS Khalid. Soft-core processors for embedded systems. In *2006 International Conference on Microelectronics*, pages 170–173. IEEE, 2006.

[16] R. Weigand and D. M. Codinachs. Single event effects in sram based fpga for space applications analysis and mitigation (dsnoc'09)., 2009.

[17] Michael J Wirthlin, Andrew M Keller, Chase McCloskey, Parker Ridd, David Lee, and Jeffrey Draper. Seu mitigation and validation of the leon3 soft processor using triple modular redundancy for space processing.

In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 205–214. ACM, 2016.

[18] Xilinx. 7 series fpgas configurable logic block. Jan. 2011 [Revised Sept.2016].

[19] YZ Xu, H Puchner, A Chatila, O Pohland, B Bruggeman, B Jin, D Radaelli, and S Daniel. Process impact on sram alpha-particle seu performance. In *2004 IEEE International Reliability Physics Symposium. Proceedings*, pages 294–299. IEEE, 2004.