Peña-Fernandez, M., Lindoso, A., Entrena, L., Garcia-Valderas, M., Philippe, S., Morilla, Y. & Martin-Holgado, P. (2018). PTM-based hybrid error-detection architecture for ARM microprocessors. *Microelectronics Reliability*, vol. 88-90, pp. 925–930.

DOI: 10.1016/j.microrel.2018.07.074

PTM-based hybrid error-detection architecture for ARM microprocessors

M. Peña-Fernandez[a], A. Lindoso[b,*], L. Entrena[b], M. Garcia-Valderas[b], S. Philippe[c], Y. Morilla[d], P. Martin-Holgado[d]

[a] *Arquimea Ingenieria SLU., Leganes, Madrid, Spain*
[b] *University Carlos III de Madrid, Avda Universidad 30, Leganes 28911, Madrid, Spain*
[c] *INP Toulouse, 6 allée Emile Monso, Toulouse, France*
[d] *CNA, University of Sevilla, CSIC, JA,Avda Tomas Alba Edison 7, Sevilla, Spain*

Abstract

This work presents a hybrid error detection architecture that uses ARM PTM trace interface to observe ARM microprocessor behaviour. The proposed approach is suitable for COTS microprocessors because it does not modify the microprocessor architecture and is able to detect errors thanks to the reuse of its trace subsystem. Validation has been performed by proton irradiation and fault injection campaigns on a Zynq AP SoC including a Cortex-A9 ARM microprocessor and an implementation of the proposed hardware monitor in programmable logic. Experimental results demonstrate that a high error detection rate can be achieved on a commercial microprocessor.

1. Introduction

Microprocessors are commonly used in a wide variety of applications, including safety-critical and high availability missions. In these applications, meeting the reliability requirements in an effective manner is a challenge. Among the multiple factors that may affect reliability, radiation-induced soft errors have the potential to cause the highest failure rate of all other reliability mechanisms combined [1]. Therefore, they are a primary concern in applications working in extreme environments, such as space, and a growing concern also at the ground level.

Although there are radiation-hardened microprocessors specifically developed for these type of environments, they are generally expensive and have high power consumption. Moreover, their performance generally lags behind commercial processors. As a consequence, there is a growing interest in the use of COTS (Commercial Off-The-Shelf) microprocessors even for space applications [2]. In this case, error detection or mitigation must be provided taking into account that the hardware cannot be modified.

Software fault-tolerance techniques [3] introduce redundancy in the code in order to detect or correct errors. These techniques have been widely studied and are the basic solution for COTS microprocessors. However, they are limited because processors contain many sensitive resources that cannot be directly accessed through software. In addition, they introduce significant performance penalties. These limitations are particularly relevant in the case of control-flow error mitigation [4].

To overcome these limitations, the use of hardware monitoring has been proposed [5]. Hardware monitoring uses an additional piece of hardware that can observe the execution flow of the processor through a suitable interface. Debug resources, which are commonly available in most microprocessors to facilitate system development and software debugging, can be reused for this purpose. These resources are useless during normal operation, so they can be reused for on-line monitoring in an inexpensive way. On the other hand, they can provide internal access to the microprocessor without disturbing it. In particular, the use of program trace interfaces has been proposed and demonstrated for soft cores [5], [6], [7]. In a soft core, it is possible to use a low-level or custom trace interface that provides

great flexibility and performance. However, in the case of commercial cores, trace interfaces are usually complex and require trace information to be decoded and synchronized for the application.

In this work we propose and evaluate a hybrid error-detection architecture for ARM processors. ARM is currently one of the most popular choices for embedded systems and supports debug and trace functions through the CoreSight™ subsystem [8]. Coresight is actually a family of IP (Intellectual Property) modules. In this paper, we focus on the Program Trace Macrocell (PTM), a CoreSight component that provides program-flow trace information. The PTM is the basic program flow trace macrocell for the ARM Cortex-A9 architecture [9].

In the proposed hybrid approach, the code is hardened for data errors, using duplication, while control-flow errors are detected by a hardware monitor attached to the PTM through the trace port. The hardware monitor continuously receives and decodes trace packets along the execution of the application program, extracts the control-flow information and checks it on-line.

Validation of the proposed hybrid architecture has been performed with fault injection and proton irradiation campaigns. Fault injection is a widely used approach to evaluate the effects of faults in an inexpensive way, but it is limited to user accessible components. Additionally, a proton irradiation campaign has been performed to test the proposed hybrid architecture in a more realistic way. Both tests provided very similar results. We show how the combination of data duplication and hardware monitoring provides a good error detection capability. We also evaluate the contribution of each part of the system to the error detection rate.

The remaining of this paper is as follows. Section two summarizes related work and introduces some concepts about hybrid architectures based on the trace interface. Section three describes the proposed hybrid architecture. Section four shows the experimental results. Finally, section five presents the conclusions of this work.

2. Related work

Microprocessor hardening techniques are usually divided into software, hardware, and hybrid techniques [3]. The type of detected errors by all these techniques is commonly divided into errors affecting control-flow and errors affecting data. Control-flow errors modify the execution flow causing the microcontroller to execute a different instruction than the one that had to be executed. Data errors affect exclusively to program data.

Software techniques modify the application software to detect or correct errors. The main advantages of software techniques are flexibility and ease of implementation. Generally, software techniques require larger execution time and increase memory usage (due to the software modifications and required additional storage for comparison information). Software techniques can be also divided into data and control-flow techniques.

Data techniques are commonly based in duplication. Data duplication consists in duplicating all variables used in a program. Original data and duplicated data must perform the same operations. During program execution, duplicated and original data must be checked. Errors are detected when a difference in both data sets is found. In [10] a set of rules are defined to modify the software for this purpose. This work achieves a very good error coverage but with a high impact in area and execution time. In order to solve these limitations, duplication can be applied at different levels, looking for a trade-off between error coverage and performance penalty. Duplication can be performed at instruction, function or even program level. Other possibilities that are present in the literature to decrease the performance and size penalties are based in reducing the number of data checkpoints or limiting the duplicated data. In [11] instead of duplicating all data, specific variable sets are duplicated. Ref. [12] evaluates the relevance of variables and applies a set of rules for selective duplication in order to reduce the impact of duplication.

The most common software control-flow techniques are based on assertions or signatures. Signature-based techniques commonly divide the program code into basic blocks. A basic block is a set of instructions with no branches except for possibly the last one. At compilation time a signature is assigned to every basic block. At execution time, the signatures are computed and checked at the end of every basic block. It must be noted that compilation time signatures require additional storage that may introduce a significant memory size penalty. Examples of these techniques are CEDA [13], ECCA [14] and YACCA [15]. Assertion-based techniques modify the code by inserting special statements (assertions) that check the data-flow correctness. In this case, error coverage can be affected by the

assertion location and also by the information included in it, so that they are application-dependent. An example of the use of assertions can be found in [16].

Hardware techniques modify the circuit architecture to harden it. A well-known example of these techniques is TMR (Triple Modular Redundancy). In the case of COTS microprocessors, the architecture is not commonly available. In addition, a new device has to be manufactured to include the hardware modifications. These drawbacks make the application of this kind of techniques unfeasible for COTS in most cases.

Alternatively, error detection in microprocessors can be accomplished by connecting additional external hardware modules to observe the system behaviour. The error coverage usually depends on the capacity of observation through the feasible connections. Several works have used this approach, proposing hardware modules [17], [18], [19] that range from simple circuits to very complex ones that could be considered similar in complexity to the observed microprocessor. These hardware modules are commonly named watchdog processors. Watchdog processors can also be classified into active and passive. Passive watchdog processors can be used to check signatures or assertions inserted in the software executed by the microprocessor. They commonly require additional memory to store the values for comparison. Active watchdog processors decrease the memory needs but increase the complexity and the required area. These processors are able to execute a simplified version of the program executed by the microprocessor. Examples of these processors are proposed in [17] and [18].

Hybrid techniques combine both software and hardware techniques taking advantage of their individual benefits. The most common approach is to apply software techniques for data-flow hardening, as data is more complex to observe externally, and use the hardware monitor to detect control-flow errors. For instance, in [20] a hardware module is used to monitor the control flow while software fault tolerance techniques are used to detect errors in the data-flow.

Microprocessors are commonly observed through memory buses or through the trace interface. A trace subsystem is commonly included in most microprocessors to support software debugging. When the debugging process is finished, this part of the circuit is not used. In [5], an extensive overview of the use of the trace interface for microprocessor observation is presented. The use of the trace subsystem for on-line monitoring was first proposed in [21] to observe a LEON3 microprocessor. In this work, several microprocessors were executing the same software at different times. During execution, signatures were generated from the available trace information. The coverage can vary depending on the selected information that is used to obtain the signatures. An extended approach was proposed in [22], where critical tasks are replicated (in the same microprocessor or in different microprocessors) and the information provided by the trace interface is compared for both executions. The comparison is accomplished by an external hardware module that computes a signature based on trace information. Other approaches have been proposed that make a more elaborated use of the trace information. In [23], a hybrid technique is proposed using the trace interface to harden the execution flow while data errors are handled with SWIFT-R technique. A new technique was proposed in [6] that compares the program flow information retrieved from two different points: the trace interface and the memory bus. This technique was able to detect all control-flow errors in a LEON3 microprocessor.

3. Hybrid architecture

3.1 Hardware monitor

In this paper we present a hardware monitor that observes the execution of an ARM Cortex-A9 core through its trace interface. The hardware monitor is capable of decoding and checking program trace information. It has been developed as an IP core that can be configured as a system peripheral. A Xilinx Zynq-7010 [25] All Programmable System-on-Chip (AP SoC) device, including a dual-core ARM Cortex-A9 processing system, has been chosen as the test platform for the proposed system. An overview of the complete system is shown in Fig. 1.

The trace interface provided by the ARM Cortex-A9 is based on the CoreSight™ technology. CoreSight [8] is a family of IP modules intended to support the needs for debug access, instruction tracing, cross-triggering and time-stamping. Some of the most common Coresight components are represented on the left side of Fig.1 as the Instrumentation Trace Macrocell (ITM), the Fabric Trace Monitor (FTM), the Funnel, or the Trace Port Interface Unit (TPIU). In this work, we focus on one

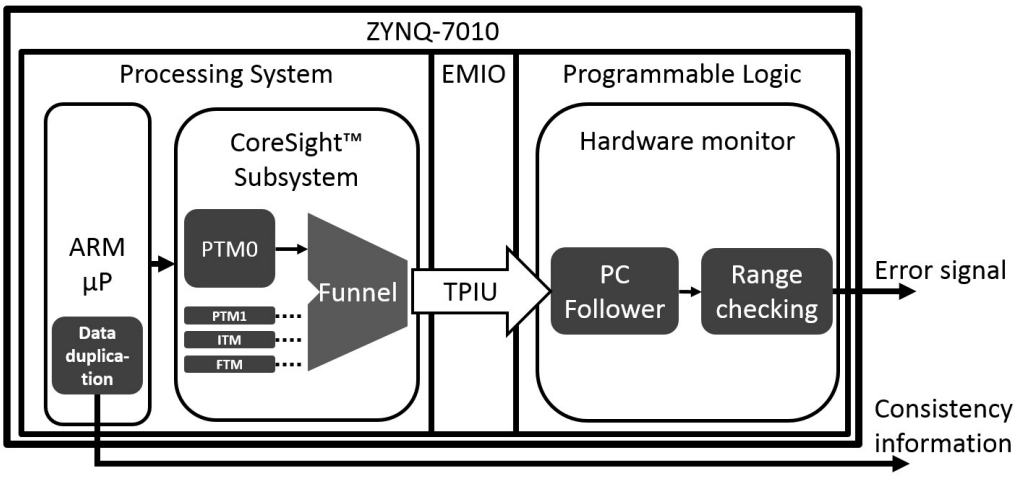

Fig. 1. Proposed system overview.

specific CoreSight component, called Program Trace Macrocell (PTM). The PTM is a real-time module that provides instruction tracing of a processor. It is a CoreSight component of the trace source class based on the ARM Program Flow Trace (PFT) architecture specification [9]. Two PTM units are provided in the Zynq-7010 AP SoC, called PTM0 and PTM1, one for each core.

The PTM produces useful information to understand the operation of the processor in a format designed to optimize bandwidth. This is achieved by generating compressed data, which contains just the minimum information required to reconstruct the processor execution flow. To enable correct interpretation of core execution, ARM PFT architecture also provides full information about exceptions, the instruction set state, security state and current Context ID of the processor. The information is formatted in packets. Each packet is composed of a variable, but bounded, number of 8-bit words. To distinguish between different packet types, the first word of each packet, called the header, must be checked and decoded. The ARM PFT architecture specification ensures a unique header for each packet type to guarantee the correct interpretation of the enclosed information. With respect to this protocol, it is important to note that all packets must be correctly identified and delimited to prevent the monitor from getting lost, regardless of their relevance for the checking process.

A hardware monitor has been developed based on the ARM PFT architecture specification to decode and extract the trace packets generated by the PTM. This monitor has been implemented in the programmable logic of a ZYNQ-7010 and connected to the ARM Processing System through the CoreSight Trace Port Interface Unit (TPIU) using Zynq EMIO (Extended Multiplexed I/O) interface. Trace information is produced by the PTM and driven through the Funnel to the TPIU, so the corresponding Funnel input must be enabled. All involved CoreSight components are configured and enabled by software during the microprocessor initialization.

During operation, the hardware monitor receives and decodes trace packets. In the ARM PFT protocol, the amount of words in each packet is variable and only by identifying the last word in one packet it is possible to identify the header of the next packet. Also, data contained in each word may be relevant to interpret the next ones. For these reasons, a pipelined architecture has been implemented to reliably extract trace information irrespective of the length of the received packets or their order. This way, each packet can be correctly identified and delimited, making the hardware monitor continuously aware of the type of packet which is currently being decoded.

Once the hardware monitor is able to identify and delimit all packet types, any further functionality can be implemented using information available in the received packets. In our application, the available information is used to obtain and monitor the Program Counter (PC) of the ARM processor. The PC value is obtained using information from three main types of packets: *I-sync* packet, *Branch Address* packet and *Waypoint Update* packet. With this method, the PC value can be updated periodically, in every waypoint.

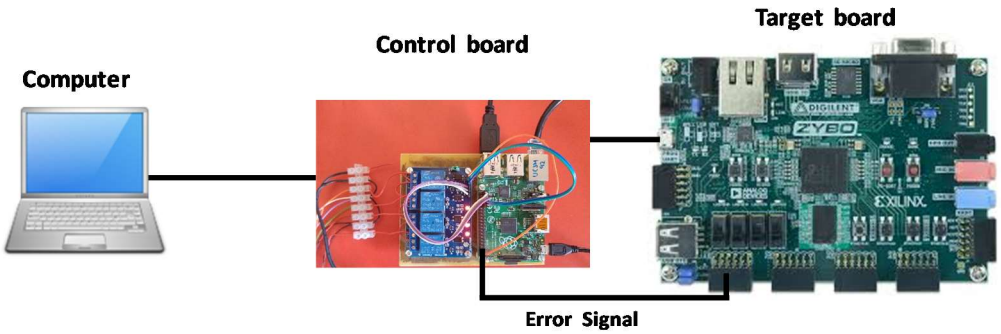

Fig. 2. Experimental setup overview.

ARM PFT architecture specification defines a waypoint as a point where an instruction might involve a change in the program flow. The described functionality is called a *PC follower* and provides updated PC value information that can then be used to determine the processor behaviour during execution and detect if it is correct or not.

To detect control-flow errors, a range checking method has been implemented. The hardware monitor has been designed to allocate up to eight configurable PC ranges, each of which can be configured through the AXI peripheral interface. These ranges have been named *confidence ranges*, and they determine allowed PC values during execution. In practice, a user must configure confidence range values with the addresses where the user application functions are stored. Any time the hardware monitor detects that actual PC value is not within any of the valid confidence ranges, an error signal is asserted.

3.2 Data error detection

The error detection capabilities provided by the hardware monitor are complemented by conventional software techniques based on data duplication. Basically, all variables are duplicated and all operations are also duplicated on the variable copies. Data consistency checks are also included in the software. They are performed just after variable modification to minimize error detection latency. As mentioned in section 2, variable duplication can be optimized to reduce the overheads. However, in the implementation used in this work all variables were duplicated for the sake of completeness.

4. Experimental results

The proposed approach has been tested with proton irradiation and fault injection campaigns. A common experimental setup was used for both experiments in order to make the results as coherent as possible. The experimental setup is described in section 4.1. Then, sections 4.2 and 4.3 describe the performed experiments and the results obtained in each case.

4.1 Experimental setup

For the experiments we used commercial boards, namely Zybo boards. Zybo contains a XC7Z010 device from Zynq-7000 AP SoC family of Xilinx, which includes a dual core ARM Cortex™-A9 processor. One single core of the device was used, running at the nominal 650 MHz clock frequency. The device also includes a Programmable Logic (PL) part which was used to implement the proposed hardware monitor.

Fig. 2 shows a picture of the experimental setup. The Device Under Test (DUT) is included in the Zybo board. The control of the experiment is performed by an external control board that collects and records all the information about the errors that occur during the experiment. The control board can also restart and reset the DUT when non-recoverable errors are observed.

All the necessary configuration data, including the boot program, the PL configuration bitstream and the application software program are stored in an SD card that is copied to OCM (On Chip Memory) when the device is turned on. As our proposed hardware monitor is located in the programmable logic of the device, it can also be affected by errors. To mitigate

these errors, the Xilinx Soft Error Mitigation (SEM) Controller IP was used. Xilinx SEM IP can detect, correct and classify SEUs in the configuration memory of the PL. During the experiments, the SEM is connected to the control board to send all the information about SEU detection, correction and classification. Errors in programmable logic that cannot be corrected by the SEM trigger a reprogramming of the device by the external control board.

We have used a matrix multiplication application software benchmark for the experiments. Following the proposed hybrid solution, the software benchmark was modified to implement data duplication, as described in section 3.2. The benchmark was compiled with Xilinx SDK environment and minimum optimization effort (-O0) in order to prevent the compiler from eliminating duplicated variables. The benchmark is running an infinite loop computing the matrix multiplication of 32x32 elements arrays. The complete software size is approximately 111 kB.

Errors were classified according to the following categories:

- Hang error: Microprocessor cannot continue normal execution and requires the system to be restarted and reconfigured. This category takes into account Hang errors that are detected only by the external control board.
- Detected Hang error (Det Hang): Hang errors that are detected by the proposed hardware monitor and the external control board.
- Detected data error (Det. SW): Software data duplication has detected a discrepancy in duplicated data. These data errors are detected by the software checks and reported to the external control board.
- Communication error (Comm): Communication between the FPGA and the external control board is experiencing a malfunctioning.

4.2 Proton irradiation

In order to test the proposed approach we have performed a proton irradiation experiment that took place in March of 2018 at CNA (Centro Nacional de Aceleradores), Spain. The experiments were performed using the external beam line installed in the 18/9 IBA compact cyclotron. The DUT was irradiated in open air with 15 MeV protons. The energy of incident protons in the silicon active area is in the order of 10 MeV, which is enough to produce events for the used technology of 28 nm without the need for thinning the devices [24]. The total fluence was $1.6 \cdot 10^{12}$ p/cm^2.

Table 1 shows the number of observed errors and their percentage with respect to the total number of errors for each error category.

Table 1
Experimental results of proton irradiation

Error type	#Errors	%Errors
Hang	7	2.30%
Comm	3	0.98%
Det. SW	236	77.38%
Det Hang	59	19.34%
Total	305	100.00%

Experimental results show that the proposed approach presents a high capacity of error detection and is able to detect 96.72% of the observed errors. Considering the errors that can be observed by the external hardware monitor, it can detect 89.39 % of the observed Hang errors.

The total cross section is $1.91 \cdot 10^{-10}$ cm^2 with a 95% confidence interval between $1.69 \cdot 10^{-10}$ cm^2 and $2.12 \cdot 10^{-10}$ cm^2. When only undetected errors are considered (categories Comm and Hang from Table 1), the cross-section reduces to $6.25 \cdot 10^{-12}$ cm^2 ($3.0 \cdot 10^{-12}$ cm^2 - $1.15 \cdot 10^{-11}$ cm^2), which is more than 30 times smaller.

4.3 Fault injection

Complementarily to the proton irradiation experiment, we tested our proposed approach with fault injection. We injected faults in the registers in the microprocessor to evaluate the microprocessor behaviour in a more detailed way. For the injection we have used the very same experimental setup that was utilized in the radiation experiments.

The implemented fault injection approach is based on the Code Emulation Upset technique [26]. This approach is summarized as follows. A timer is configured to trigger an interrupt at a random instant. Upon interruption, the full set of registers of the ARM microprocessor is saved on the stack, so that they are available for fault injection. Then, a bit-flip is injected

in a randomly selected bit of one of the registers. When the processor returns from the interrupt, the registers are restored from the stack and the single bit injected fault becomes effective. The execution is resumed and is let running for several iterations of the tested application software.

A preliminary run of the application software is used to measure its execution time, which is used as the maximum range for random generation of injection instants. The injected register and bit are also randomly selected. Random seeds are generated externally and provided to the device when it is restarted in order to ensure that random values are generated without bias.

The ARM processor contains a large set of registers. In particular, it uses banked copies of some registers, with the current register selected by the execution mode. In addition, the Single-Instruction Multiple Data (SIMD) and floating-point coprocessors have their own set of registers, which can also be saved in the stack. Faults can be injected in any register using our approach. However, fault injection was performed only on the ARM core registers at the application level view to avoid unnecessarily injecting faults in registers which were not used. Some registers need to be treated in a specific way considering their behaviour. Especially, fault injection in the Program Counter (PC) was actually implemented through the Link Register (LR), because the PC takes the contents of the LR upon return from interrupt.

The results of the fault injection campaign are summarized in Table 2. We injected a total of 53,488 faults, of which 12,040 (23.46%) produced observable errors. The proposed approach detected 95.94% of the errors with a 95% confidence interval of ±1.71%. The external monitor detected 89.65% of the Hang errors, with a 95% confidence interval of ±2.65%. These results are in line with those obtained in the proton irradiation experiment, which are within the calculated 95% confidence intervals. The only significant difference is that Hang errors occurred more frequently under fault injection, which is due to the narrower focus of the fault injection experiment. However, the hardware monitor was able to detect a similar amount of errors in both experiments.

Fig. 3 shows a comparison of errors on a register basis. Fault injection was performed in the complete register file, but only a subset of registers was really used due to the compilation options. For clarity, only the registers that produce at least one error are reported. The frame pointer (FP) and the program counter (PC) are the most critical registers and provoke an error for almost every bit-flip injected in them. For the rest of the registers, error sensitivity may vary depending on their usage.

Table 2
Experimental results of fault injection

Error type	#Errors	%Errors
Hang	509	4.06%
Comm	0	0.00%
Det. SW	7,631	60.81%
Det Hang	4,409	35.13%
Total	12,549	100.00%

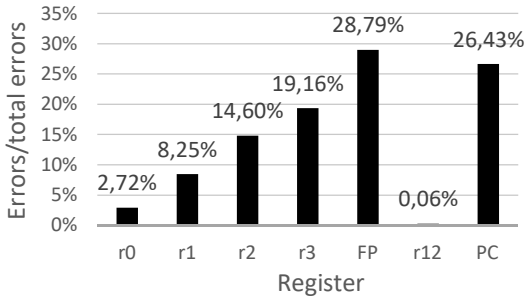

Fig. 3. Errors by register.

5. Conclusions and future work

This work presents a hybrid architecture that can monitor ARM microprocessor execution thanks to the observation of the program flow trace provided by PTM trace module. This solution presents a feasible and non-intrusive way of detecting errors in ARM-based COTS with reduced impact in area. Experimental results demonstrate the high error detection capabilities of the proposed approach.

The proposed approach has been tested with proton irradiation and fault injection. Notably, the results of both tests were very similar, although fault injection was limited to the ARM core registers. Future work is oriented to enhance the error detection capabilities based on the trace information.

Acknowledgements

This work was supported in part by the Spanish Ministry of Economy and Competitiveness under project ESP2015-68245-C4-1-P and by the Community of Madrid under grant IND2017/TIC-7776.

References

[1] R. C. Baumann, Radiation-induced soft errors in advanced semiconductor technologies, IEEE Trans. on Device and Materials Rel., vol. 5, no. 3 (2005), pp. 305-316.

[2] L. Entrena, M. Portela, A. Lindoso, M. García-Valderas, L. Mengíbar, L. Parra, J. A. Pulido, A. Latorre, Flexible approaches to fault-tolerant microprocessors for space aplications, Proc. Data Systems In Aerospace (DASIA), ESA Special Publication SP-732, May 2015.

[3] M. Nicolaidis, Soft errors in modern electronic systems, (Ed.), Springer, 2011.

[4] J. R. Azambuja, S. Pagliarini, L. Rosa, and F. L. Kastensmidt, Exploring the limitations of software-only techniques in SEE detection coverage, Journal of Electronic Testing, no. 27, (2011), pp. 541–550.

[5] L. Entrena, A. Lindoso, M. Portela-García, L. Parra, B. Du, M. Sonza Reorda, L. Sterpone, Fault-tolerance techniques for soft-core processors using the Trace Interface, In "FPGAs and Parallel Architectures for Aerospace Applications. Soft Errors and Fault-Tolerant Design", Springer, 2015.

[6] L. Parra, A. Lindoso, M. Portela-Garcia, L. Entrena, B. Du, M. Sonza Reorda, L. Sterpone, A New Hybrid Nonintrusive Error-Detection Technique Using Dual Control-Flow Monitoring, IEEE Transactions on Nuclear Science, vol. 61, no. 6 (2014), pp. 3236-3243.

[7] B. Du, E. Sanchez, M. S. Reorda, J. P. Acle, A. Tsertov. FPGA-controlled PCBA power-on self-test using processor's debug features, IEEE Int. Symp. on Design and Diagnostics of Electronic Circuits & Systems (DDECS), 2016.

[8] ARM Inc., CoreSight Components – Technical Reference Manual, 2009.

[9] ARM Inc., CoreSight Program Flow Trace Architecture Specification, 2011.

[10] P. Cheynet, B. Nicolescu, R. Velazco, M. Rebaudengo, M. Sonza Reorda, M. Violante, Experimentally evaluating an automatic approach for generating safety-critical software with respect to transient errors, IEEE Transactions on Nuclear Science, vol. 47, no. 6 (2000), pp. 2231–2236.

[11] A. Benso, S. Chiusano, P. Prinetto, L. Tagliaferri, A C/C++ source-to-source compiler for dependable applications, IEEE International Conference on Dependable Systems and Networks, 2000, pp. 71-78.

[12] B. Nicolescu, R. Velazco, Detecting soft errors by a purely software approach: method, tools and experimental results, Design, Automation and Test in Europe Conference, 2003, pp. 57 – 62.

[13] R. Vemu, J.A. Abraham, CEDA: Control-Flow Error Detection through Assertions, Proc. 12th IEEE International On-Line Testing Symposium (IOLTS), 2006, pp. 151-158.

[14] V.S.S. Nair, H. Kim, N. Krishnamurthy, J.A. Abraham, Design and Evaluation of Automated High-Level Checks for Signal Processing Applications, Proc. SPIE Advanced Algorithms and Architectures for Signal Processing Conf., Ago. 1996.

[15] O. Goloubeva, M. Rebaudengo, M. S. Reorda, M. Violante, Soft-error detection using control flow assertions, 18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 2003, pp. 581–588.

[16] M. Hiller, Executable assertions for detecting data errors in embedded control systems, Proceedings of the IEEE international conference on dependable systems and networks, 2000, pp 24–33.

[17] T. Michel, R. Leveugle, G. Saucier, A New Approach to Control Flow Checking Without Program Modification, Proc. 21th International Symposium on Fault-Tolerant Computing (FTCS-21), 1991, pp. 334-341.

[18] S. Bergaoui, R. Leveugle, ISDM: An Improved Control Flow Checking Approach with Disjoint Signature Monitoring, Proc. 24th Conference on Design of Circuits and Integrated Systems (DCIS), 2009.

[19] A. Benso, S. Di Carlo, G. Di Natale, P. Prinetto, A Watchdog Processor to Detect Data and Control Flow Errors, 9th IEEE International On-Line Testing Symposium, 2003, pp. 144-148.

[20] J. R. Azambuja, M. Altieri, J. Becker, F. L. Kastensmidt, HETA: Hybrid Error-Detection Technique Using Assertions, IEEE Transactions on Nuclear Science, vol. 60, no. 4 (2013), pp. 2805-2812.

[21] M. Grosso, M. Sonza Reorda, M. Portela-Garcia, M. Garcia-Valderas,C. Lopez-Ongil, L. Entrena, An on-line fault detection technique based on embedded debug features, Proc.16th IEEE On-LineTesting Symposium, 2010, pp. 167-172.

[22] M. Portela-Garcia,M.Grosso, M. Gallardo-Campos,M. Sonza Reorda, L. Entrena, M. Garcia-Valderas, C. Lopez-Ongil, On the use of embedded debug features for permanent and transient fault resilience in microprocessors, Microprocessors Microsystems, vol. 36, no. 5. (2012), pp. 334–343.

[23] L. Parra, A. Lindoso, M. Portela, L. Entrena, F. Restrepo-Calle, S. Cuenca-Asensi, A. Martínez-Álvarez, Efficient Mitigation of Data and Control Flow Errors in Microprocessors, IEEE Transactions on Nuclear Science (TNS), vol. 61, no.4 (2014), pp. 1590-1596.

[24] A. Lindoso, M. García-Valderas, L. Entrena, Y.

Morilla and P. Martín-Holgado, Evaluation of the suitability of NEON SIMD microprocessor extensions under proton irradiation, IEEE Transactions on Nuclear Science, 2018 (in press).
[25] Xilinx Inc., Zynq-7000 All Programmable SoC: Technical Reference Manual, UG585, 2016.
[26] R. Velazco, S. Rezgui and R. Ecoffet, Predicting error rate for microprocessor-based digital architectures through C.E.U. (Code Emulating Upsets) injection, IEEE Transactions on Nuclear Science, vol. 47, no. 6 (2000) pp. 2405-2411.