Investigating the Inherent Soft Error Resilience of Embedded Applications by Full-System Simulation

(Invited Paper)

Uzair Sharif, Daniel Mueller-Gritschneder, Ulf Schlichtmann
Chair of Electronic Design Automation
Technical University of Munich, Germany
{uzair.sharif,daniel.mueller,ulf.schlichtmann}@tum.de

***Abstract*—It has long been acknowledged that some applications feature inherent resilience against soft errors, e.g., the impact of soft errors on multimedia applications is often non-visible to humans. In this paper we investigate the inherent resilience of two typical embedded applications using a case study of a control system and a robot arm. Both studies were enabled by our mixed-mode fault injection simulator ETISS-ML, which allows RTL-accurate fault injection while being able to simulate very long scenarios, e.g. robot movements of several seconds. Our results indicate that full simulation of the embedded system and its environment are required to classify whether the system can tolerate the impact of a soft error. This is due to the fact that it is hard to predict the impact of a certain output deviation without investigating the change in the system behavior taking into account the control loop. Based on this classification method we hope to be able to exploit this resilience for lowering the cost of error detection mechanisms in future research.**

***Index Terms*—Soft error resilience, silent data corruption, application resilience, safety critical embedded systems**

I. Introduction

Embedded systems are increasingly being deployed in safety-critical domains. Safety-critical systems are characterized as systems whose failure could cause catastrophic damage to the surrounding environment [1]. Hence, designing them for failure-free operation is a top priority. This entails designing dedicated safety mechanisms, in addition to functional components, which inadvertently leads to higher overhead costs. Tailoring embedded-system design for safety-critical domains, hence, poses a major technical challenge: to provide adequate operational safety while still ensuring power and cost efficiency. To achieve this, products in industries as diverse as automotive, transportation, medical etc. are opting for simpler commercial-off-the-shelf (COTS) platforms over custom hardened ones for meeting their computing demands. This is mainly attributed to embedded systems' ever increasing computational capabilities while still retaining their characteristic benefits in terms of cost efficiency.

Radiation-induced *soft errors* [2] are considered to be a major source of run time errors that could cause a *safety-critical embedded system* (SCES) to fail. Furthermore, the soft error rate is expected to rise with emerging HW architectures. This is due to advanced technology scaling (in terms of device geometries, operating voltages etc.) being employed to realize faster power-efficient implementations. Soft errors manifest themselves at system level as either *detected, but unrecoverable errors* (DUEs) or *silent data corruptions* (SDCs) [3]. Most COTS processors are equipped to handle DUEs. For example, a watchdog timer (WDT), a commonplace safety mechanism in these processors, can detect error scenarios that lead to SW being inactive for extended periods of time. Similarly, a memory protection unit can detect illegal memory accesses at run-time, thereby alerting the CPU via an exception. Numerous studies have already been conducted to investigate the impact of SDCs (e.g. [4], [5]). This has helped to devise various solutions to tackle SDCs. Among these, *software implemented HW fault tolerance* [6] (SIHFT) is highly attractive for SCES, due to minimal HW requirements [7]. The key design philosophy behind SIHFT has, until now, been to yield maximum, if not perfect, coverage of SDCs with only a secondary focus on incurred overheads. While selective SIHFT variants have been suggested to trade-off SDC coverage with run-time overheads [8], [9], the SDC protection offered by them is deemed insufficient for safety-critical operation.

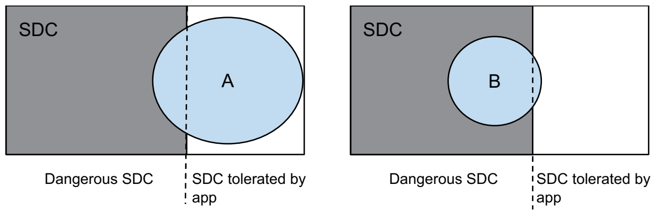

Fig. 1. SDC-coverage map for different safety mechanisms

Many embedded applications are designed under *soft-computing* [10] semantics, in order to robustly handle unexpected conditions or noise when operating in harsh environments. They don't require numerically perfect computations for carrying out their functionality, but instead are able to tolerate a certain amount of corruption in their data. Expressed another way, these applications offer a certain amount of inherent resilience. Our core insight is that by leveraging such *application-level resilience*, more optimal design points can be generated than those possible by relying solely on state of the art SIHFT solutions. To further motivate this, we present a soft error resilience comparison (in terms of SDC coverage) between safety mechanisms *A* and *B* for a hypothetical SCES scenario. Here, the shaded (gray) region represents the potentially dangerous (failure-causing) SDCs, while the unshaded (white) region reflects SDCs that are considered safe as, due to application-level resilience, they do not cause failures. Remaining SDCs are depicted as being covered via employed safety mechanisms (like WDT, SIHFTs etc.). As can be seen, even though *A* scores higher in terms of overall SDC coverage, *B* is better in leveraging the inherent resilience of the application because *B* covers more of the potentially dangerous SDCs while *A* instead covers mostly safe SDCs.

The concept of exploiting application resilience to improve the design space exploration process for a specific domain has been considered in the literature. For example, it has been exploited in devising energy-efficient HW accelerators specifically for machine-learning workloads [11], in developing optimized GPU-compute kernels for image-processing applications [12], in improving yield of complex VLSI chips for multimedia processing [13] etc. From a fault-tolerance perspective, Thomas et al. [14] were among the first to formally express application resilience by introducing the notion of egregious data corruptions. Essentially, egregious here refers to severe forms of SDCs to distinguish them from the rest of the SDCs. Instead of conventional focus on SDC-coverage, authors in [14] proposed to focus exclusively on such potentially dangerous SDCs to generate more optimal design points. However, they only demonstrated their method by analyzing benchmark programs, which are not of interest in the context of SCES operation.

We extend their concept of egregious SDCs to cover complex, realistic embedded scenarios as well. For this we consider current safety standards such as ISO26262. In these standards error impact is categorized as either potentially dangerous or safe. A potentially dangerous error may violate a so-called safety goal. When the application running on a COTS platform is unknown, any SDC needs to be categorized as potentially dangerous. In contrast, when the application is known, then the error impact can be simulated and its effect can be categorized based on its impact on the system behavior. Our future goal is to demonstrate that distinguishing between safe and dangerous SDCs will allow to reduce the cost of protecting SCESs against failures by considering that no protection against safe SDCs is required. The key point is that one needs to simulate the system behavior to investigate the application's resilience. Just looking at the output deviation is insufficient. This is demonstrated by two case studies.

II. Sources of Application's Inherent Resilience

Almost any embedded system can be modeled as a functional entity that continuously senses the information from its surrounding environment (inputs), processes it and afterwards actuates to control some aspects of the environment (outputs). Thus, any such system can entirely be (informally) described by a sense-compute-actuate relationship forming a closed loop with the physics of its environment (plant), herein referred to as system-level behavior. An SCES specifies additional safety goals (as part of the safety specification) that are expressed in terms of this system behavior. We formulate the application's inherent resilience of a given SCES as: *the amount of deviation in its system-level behavior that it can tolerate without violating any of its safety goals.*

Soft-error-induced SDCs are at first glance all potentially dangerous as they may corrupt system inputs, outputs or system states. Based on this corruption, the system may violate a safety goal due to unpredictable actuation. However, taking application-resilience into consideration, some SDCs could be deemed safe. This is motivated by the fact that there exist also other sources of uncertainties in these systems e.g., noise on the sensors may lead to different input-output behaviors. Hence, these systems must be designed to not being reliant

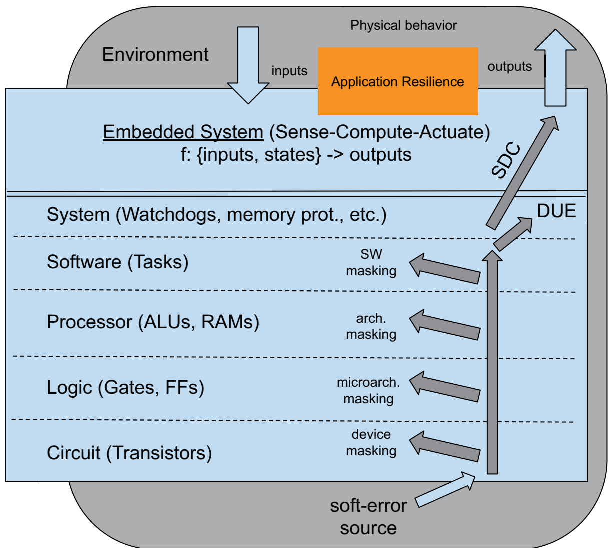

Fig. 2. Application resilience as system-level masking

on numerically perfect computations in carrying out their functionality, but instead they must be able to tolerate a certain amount of noise. As long as the corruption of an SDC is similar to such noise effects, the noise tolerance of the system will assure it is safe.

Another major source of embedded applications is the periodic execution of sense-compute-actuate tasks. When an SDC leads to a corrupt output in the actuation, the system behavior may deviate from the error-free behavior. Yet, in the next sense step, this deviation is detected and the system corrects for it. Sometimes the software just overwrites the faulty command with a correct new one in the next step. For example, when an SDC leads to a robot joint rotating in the wrong direction, the sensors in the step motors would report that the step counts are done in the incorrect direction. In the next control step, the direction would be adapted, if the software is designed accordingly. This may lead to small, yet insignificant deviations from the error-free behavior.

We further illustrate this idea in Fig. 2 which shows an abstract system representation of an SCES working within its environment. Not all faults propagate to the outputs due to a variety of masking effects at different layers. Additionally, common safety mechanisms such as watchdogs or memory detection would detect and handle DUEs. The errors that propagate to the system outputs without being detected are classified as SDCs. As highlighted, we can conceptualize application resilience at the highest abstraction layer, as an effect that has to take the sense-compute-actuate loop of the embedded system as well as the physical behavior of its environment into account. Based on the safety goals, the SDC can be potentially dangerous or safe (tolerated due to inherent resilience). Any design hardening mechanism (e.g. SIHFTs) should target the potentially dangerous SDCs rather than the entire SDC set to minimize the cost of providing fault protection.

III. Investigating an Application's Inherent Resilience

To investigate a given SCES's inherent resilience in a generic and systematic way, we propose to carry out fault injection experiments combined with full-system simulation. Full system simulation here means that we do not only simulate the embedded system and its software but also the physical behavior of the system. This is essential as SDC impact cannot always be classified from just the embedded software outputs. For example, an SDC could be considered safe when the corrupted value written by the software to an actuation output differs less than 10% from the correct value. Yet, if such a deviation occurs continuously for several actuation steps, even such a "small" disturbance could become potentially dangerous. Hence, our approach is to inject the fault and trace the system's physical behavior, e.g., the trajectory of the movement of a robot in space. Based on the definition of the inherent resilience, we compare the error-free trajectory with the trajectory generated by the software under the influence of the SDC. The deviation is classified based on the pre-defined safety goals. As long as the trajectory is deemed safe in terms of the safety goals, the SDC is classified as safe. This of course can also depend on the mission profiles. Hence, in order to result in meaningful statistics, the simulated scenarios should represent the real use cases later executed by the embedded system. This approach enables accurate investigation of the impact of each of observed SDC on the system-level behavior with respect to the pre-defined safety goals.

To illustrate this setup, we introduce two separate case-studies from the embedded domain and show their inherent resilience potential by applying our method.

IV. Case Studies

We use SystemC / TLM modeling to implement virtual prototypes (VPs) of the embedded systems in order to simulate them in a full-system manner. The VPs are centered around our ETISS-ML simulator [15] that provides the capability to perform RTL-accurate fault injection experiments within reasonable time. In this paper, we mainly focus on system-level aspects of these VPs. Interested readers are referred to prior works [15], [16] for further details regarding the VP setup.

A. Adaptive Cruise Control (ACC) System

System Description. We simulate a road scene involving two cars. In the chosen mission profile, both cars are set up to be moving at $20\,\mathrm{m/s}$ in a particular direction with an initial distance of $50\,\mathrm{m}$ between them. The ACC is configured to drive its subject car at uniform speed while keeping a safe distance of $40\,\mathrm{m}$ from the preceding car. It comprises an engine control unit (ECU), a throttle-actuator, and radar and speed sensors as key components. The radar provides updates on relative distance from the preceding car, while the speed-sensor is used to measure the subject car's speed at a given point in time. The sensors are configured in the ECU's firmware to provide updated samples periodically in time via interrupts. This sensory information is then used as input by a fault-tolerant PI controller [17] task to send appropriate control commands to the throttle (via the actuator), in order to accelerate or decelerate the car as needed. The control is updated every $10\mathrm{ms}$. This is a motivational and very simplified setup, in which the ACC system relies on a simple PI control law and the physics of the two cars are simulated with a linear plant model. The real system behavior would differ significantly due to the complex physics of the cars. Additionally, the following car could use also the brakes to control the distance, especially in emergency situations. Yet, overall this setup gives an idea of the inherent resilience of linear control tasks, which are still used commonplace in many embedded systems.

We study soft error resilience of the ACC system that is deployed inside the following car. The entire scene is simulated for $15\,\mathrm{s}$.

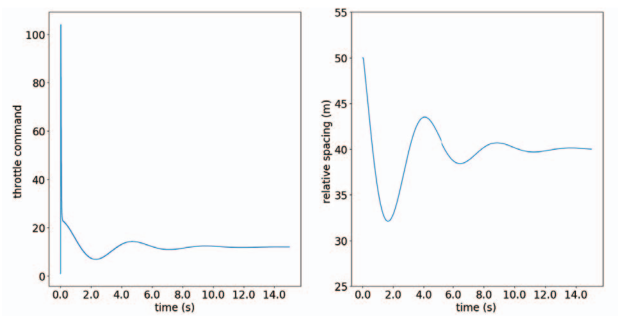

Fig. 3. ACC: Output and system response (Fault free reference)

System-level Behavior. Fig. 3 plots the throttle commands (system outputs) along with the distance between the two cars (system behavior) observed over time. As can be seen, the spacing (right plot) varies from $50\mathrm{m}$ to $30\,\mathrm{m}$ in the ACC's transient phase, with the typical transient behavior of a linear PI control. Afterwards, it quickly converges to the desired spacing of $40\,\mathrm{m}$. Likewise, the throttle plot (on the left) depicts the controller being highly aggressive at the start with its actuation. Over time the controller's actuation saturates.

We define the system's safety goal in terms of the physical behavior. As long as the two cars either get too close or drift too far away from each other, the system is deemed to have compromised safety.

Impact of SDCs. We carry out an ETISS-ML fault-injection campaign by injecting random RTL *bitflips* within the CPU pipeline of the ECU. Bitflips can result not only from radiation, but also from other fault sources such as crosstalk or noise [18]. The FI targets the initial phase $(0-3\mathrm{s})$ of the scenario. Here, an SDC is reported if at any control-step, the throttle command (i.e. system output) is found to be different from expected (fault-free) scenario.

Afterwards, we investigate the reported SDC scenarios to study their impact on the system-level behavior. The results are plotted in Fig. 4 and Fig. 5. Here again we plot the system's output response on the left and its behavior response on the right. As can be seen, the distance perturbations in Fig. 4 are much smaller than in Fig. 5 and in spite of deviation from the fault-free behavior, the system is still able to ensure safe distance between the cars. This is possible due to inherent resilience in the control algorithm to tackle noise. Fig. 5,

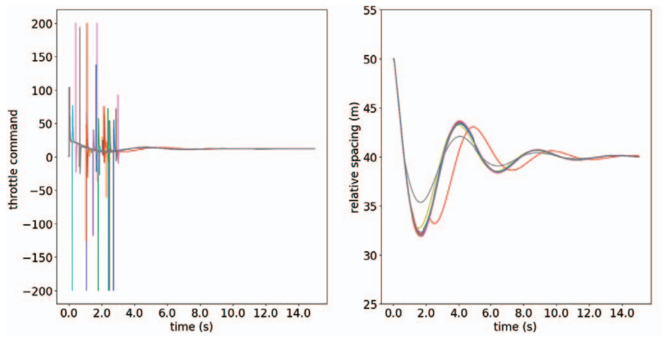

Fig. 4. ACC: Output and system response over time (Safe SDC)

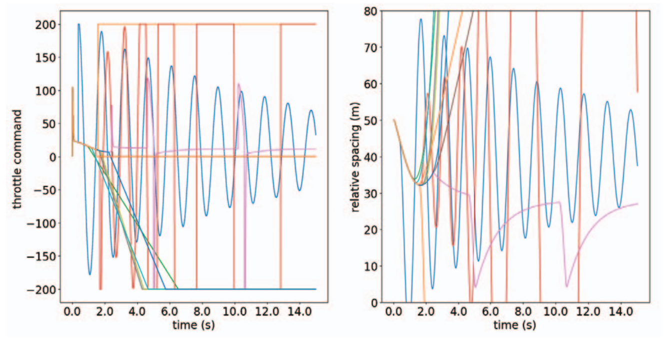

Fig. 5. ACC: Output and system response over time (Dangerous SDC)

however, depicts more severe forms of SDCs that lead to unstable control responses, resulting in a violation of the specified safety goals. For example, scenarios in which the relative spacing goes below $0\,\mathrm{m}$ represent a crash leading to injuries or even fatalities in the worst case. Naturally, these scenarios show very high acceleration due to the linear plant model, which would not be possible considering real physics. Yet, clearly the control is de-stabilized leading to an uncontrolled situation. We find roughly half of the observed SDCs to be tolerated by the application, thus demonstrating sufficient amount of inherent resilience. We expect a wide variety of embedded linear control applications to show similar resilience characteristics.

Revisiting Fig. 4, we see large deviations in throttle response even when SDCs are found to be safe. In other words, classifying SDCs into safe and dangerous classes based on the throttle response, as per related work, would have resulted in a much more conservative estimate of the application resilience. Thus, our strategy for quantifying a given system's inherent resilience based on its physical behavior rather than its output response, is further motivated.

B. Robot Arm System

System Description. We simulate a second case study of an embedded industrial system, which controls a *robot arm* as depicted in Fig. 6. The robot arm is further equipped with motion sensors and drive motors at each of its four joints. The controller (implemented as firmware) on an industrial micro-controller estimates the robot arm's current orientation via sensory information at periodic steps. It executes a simple motor control at each such step. The overall goal is to reach a sequence of given positions in 3D space starting from an

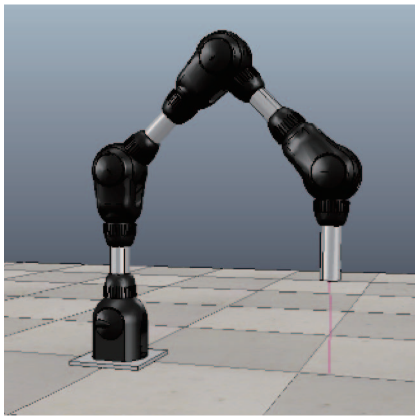

Fig. 6. A (virtual) robot arm

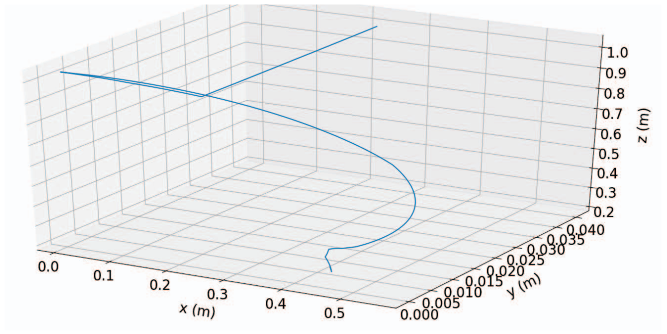

Fig. 7. Robot arm: Trajectory in 3D Space (Error free reference run)

arbitrary initial location. For this, the robot control task has two phases. First it computes the inverse kinematics of the robot arm to derive the joint positions from the 3D position. In the move phase it moves to the computed joint position by controlling the motors. We simulate the movement scenario for $20\,\mathrm{s}$.

System-level Behavior. Fig. 7 plots the 3D trajectory of the robot arm in space for the fault-free scenario. As shown, the robot arm first calibrates itself (goes to origin of its frame-of-reference). Afterwards, it computes its final (desired) orientation in order to reach the given 3D position solving the inverse kinematic equations. Finally, it executes a move phase (D curve in plot) to attain the target orientation by driving the joint motors accordingly. Here, we define the safety of the system to be compromised, if there is a significant deviation from the trajectory as this could lead to (a) possible collisions with objects in close proximity and / or (b) abrupt jerky motions of the robot arm causing self-damage.

Impact of SDCs. We study the soft error susceptibility of the robot arm controller using ETISS-ML. Here, we inject random RTL bitflips inside the CPU pipeline during the move phase of the simulated scenario. An SDC is reported for FI experiments in which the sequence of motor actuation commands differs from the one in the fault-free case.

Similar to the ACC study earlier, we also observe that not all SDCs have dangerous effects on the robot arm's system-level behavior. However, the fraction of dangerous SDCs is found to be significantly less (roughly by $10\mathrm{x}$). The corresponding

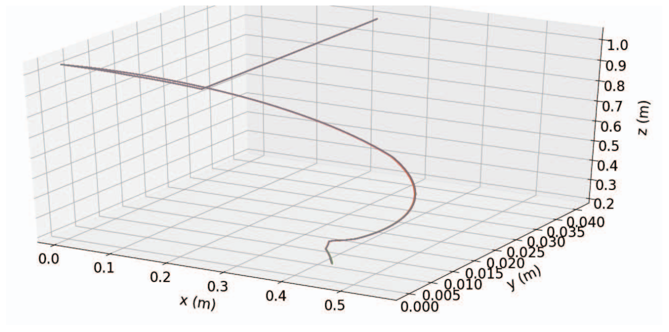

Fig. 8. Robot arm: Trajectory in 3D Space (For Safe SDC scenarios)

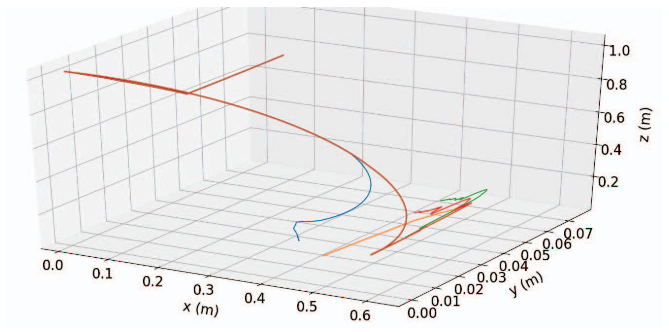

Fig. 9. Robot arm: Trajectory in 3D Space (For Dangerous SDC scenarios)

trajectory plots for safe and dangerous SDCs are shown in Fig. 8 and Fig. 9 respectively. Notice however that as opposed to the ACC case-study (Fig. 4), we see almost negligible deviations for safe SDC scenarios here.

The robot arm's better inherent resilience could well be explained by its firmware having no explicit state during the move phase except the final position computed in the previous inverse kinematics phase. Due to this, a soft error's corruption often affects at most one iteration of the motor control, with future motor control commands free from its effects. If the movement changes within one iteration, it is corrected in the next step as it is overwritten by a correct motor control setting. In contrast, the ACC firmware possesses certain state for the integral part of the PI control algorithm. A soft error corrupting this state can potentially impact several control steps. We believe this causes the higher soft error vulnerability.

V. Conclusion

In this paper, we investigated the inherent resilience potential offered by realistic embedded applications in order to reduce the fault tolerance overhead costs when deploying them in a safety critical setting. We presented an improved method for quantifying the application resilience based on analyzing the impact of SDCs on the overall system behavior, thereby motivating the need to combine fault injection simulation frameworks with full-system simulation. We demonstrated the benefits of our approach by applying them on two separate representative case studies from the safety-critical embedded domain.

Acknowledgment

This work has been partially supported by the German Federal Ministry of Education and Research (BMBF) in the project SAFE4I under grant 01IS17032.

References

[1] J. C. Knight. Safety critical systems: challenges and directions. In *Proceedings of the 24th International Conference on Software Engineering. ICSE 2002*, pages 547–550, May 2002.

[2] Norbert Seifert. Radiation-induced soft errors: A chip-level modeling perspective. *Foundations and Trends in Electronic Design Automation*, 4(23):99–221, 2010.

[3] S. Rehman, M. Shafique, and J. Henkel. *Reliable Software for Unreliable Hardware: A Cross Layer Perspective*. Springer International Publishing, 2016.

[4] S. S. Mukherjee, J. Emer, and S. K. Reinhardt. The soft error problem: an architectural perspective. In *11th International Symposium on High-Performance Computer Architecture*, pages 243–247, Feb 2005.

[5] N. J. Wang, J. Quek, T. M. Rafacz, and S. J. Patel. Characterizing the effects of transient faults on a high-performance processor pipeline. In *International Conference on Dependable Systems and Networks*, pages 61–70, June 2004.

[6] Nahmsuk Oh. *Software Implemented Hardware Fault Tolerance*. PhD thesis, Stanford, CA, USA, 2001. AAI3000076.

[7] E. Chielle, F. Rosa, G. S. Rodrigues, L. A. Tambara, J. Tonfat, E. Macchione, F. Aguirre, N. Added, N. Medina, V. Aguiar, M. A. G. Silveira, L. Ost, R. Reis, S. Cuenca-Asensi, and F. L. Kastensmidt. Reliability on arm processors against soft errors through sihft techniques. *IEEE Transactions on Nuclear Science*, 63(4):2208–2216, Aug 2016.

[8] G. A. Reis, J. Chang, D. I. August, R. Cohn, and S. S. Mukherjee. Configurable transient fault detection via dynamic binary translation. In *2nd Workshop on Architectural Reliability*, 2006.

[9] S. Rehman, M. Shafique, P. V. Aceituno, F. Kriebel, J. Chen, and J. Henkel. Leveraging variable function resilience for selective software reliability on unreliable hardware. In *2013 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1759–1764, March 2013.

[10] Lotfi A. Zadeh. Soft computing and fuzzy logic. *IEEE Softw.*, 11(6):48–56, November 1994.

[11] Z. Du, K. Palem, A. Lingamneni, O. Temam, Y. Chen, and C. Wu. Leveraging the error resilience of machine-learning applications for designing highly energy efficient accelerators. In *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 201–206, Jan 2014.

[12] M. Samadi, J. Lee, D. A. Jamshidi, A. Hormati, and S. Mahlke. Sage: Self-tuning approximation for graphics engines. In *2013 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 13–24, Dec 2013.

[13] M. A. Breuer. Multi-media applications and imprecise computation. In *8th Euromicro Conference on Digital System Design (DSD'05)*, pages 2–7, Aug 2005.

[14] A. Thomas and K. Pattabiraman. Error detector placement for soft computing applications. *ACM Trans. Embed. Comput. Syst.*, 15(1):8:1–8:25, January 2016.

[15] D. Mueller-Gritschneder, U. Sharif, and U. Schlichtmann. Performance and accuracy in soft-error resilience evaluation using the multi-level processor simulator ETISS-ML. In *Proceedings of the International Conference on Computer-Aided Design*, ICCAD '18, pages 127:1–127:8, November 2018.

[16] D. Mueller-Gritschneder, M. Dittrich, J. Weinzierl, E. Cheng, S. Mitra, and U. Schlichtmann. ETISS-ML: A multi-level instruction set simulator with rtl-level fault injection support for the evaluation of cross-layer resiliency techniques. In *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 609–612, March 2018.

[17] D. Goswami, D. Mueller-Gritschneder, T. Basten, U. Schlichtmann, and S. Chakraborty. Fault-tolerant embedded control systems for unreliable hardware. In *2014 International Symposium on Integrated Circuits (ISIC)*, pages 464–467, Dec 2014.

[18] A. Herkersdorf, H. Aliee, M. Engel, M. Glass, C. Gimmler-Dumont, J. Henkel, V. B. Kleeberger, M. A. Kochte, J. M. Kuehn, D. Mueller-Gritschneder, S. R. Nassif, H. Rauchfuss, W. Rosenstiel, U. Schlichtmann, M. Shafique, M. B. Tahoori, J. Teich, N. Wehn, C. Weis, and H.-J. Wunderlich. Resilience articulation point (RAP): Cross-layer dependability modeling for nanometer system-on-chip resilience. *Microelectronics Reliability*, 54(6-7):1066–1074, 2014.