View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by UPCommons. Portal del coneixement obert de la UPC

Modeling RTL Fault Models Behavior to Increase the Confidence on TSIM-based Fault Injection

Jaime Espinosa†, Carles Hernandez‡, Jaume Abella‡
†Universitat Politècnica de València
‡Barcelona Supercomputing Center (BSC-CNS)

***Abstract*—Future high-performance safety-relevant applications require microcontrollers delivering higher performance than the existing certified ones. However, means for assessing their dependability are needed so that they can be certified against safety critical certification standars (e.g ISO26262). Dependability assessment analyses performed at high level of abstraction inject single faults to investigate the effects these have in the system. In this work we show that single faults do not comprise the whole picture, due to fault multiplicities and reactivations. Later we prove that, by injecting complex fault models that consider multiplicities and reactivations in higher levels of abstraction, results are substantially different, thus indicating that a change in the methodology is needed.**

I. Introduction

Technology scaling has allowed processor's semiconductors industry achieving performance numbers beyond Gigaflops with reduced power budget by including a vast number of computational nodes in the same chip [1]. In the safety critical domain the huge amount of available processing power is expected to fullfil the high demands for performance guarantees of new applications like autonomous driving systems [10]. However, hardware systems targeting safety critical applications need to go through a certification process step validating its compliance with the standards [9] [5].

The need for a thorough verification and test process that certification standards poses to hardware systems may preclude the use of complex hardware platforms in the context of critical applications. Currently, the verification and test process takes between 50% and 70% of the design effort for a simple microcontroller [8]. Therefore, if high-complexity processors are to be considered for highly critical applications, like ASIL-D in the automotive domain, new methodologies and tools for the robustness verification step have to be devised [2].

Safety-critical systems industry uses simulation-based verification as a way to reduce the costs associated with the verification and validation of complex designs. Simulation-based verification allows reducing verification and validation costs as design threats and certification mismatches can be detected before manufacturing. Simulation-based robustness verification is typically carried out at RTL and gate-levels by performing extensive fault-injection campaigns that require huge computation effort to achieve meaningful results. Further increasing the level of abstraction of simulation-based fault injection will allow reducing verification costs of current designs and affording the simulation effort of high-complexity processor designs. In the case of microcontrollers, the instruction set simulator (a.k.a. TSIM) has been regarded as a potential candidate to carry out the robustness verification at a high level of abstraction. However, for TSIM's to be used in the robustness verification process it must be proven that by leveraging them accurate results can be achieved.

Several recent works [7], [22] have been carried out showing the potential and limitations of TSIM-based fault injection. In this paper, we tackle the problem of increasing the confidence on TSIM-based fault injections from a different angle. Rather than focusing on correlating the result of TSIM fault-injections with higher-accuracy fault-injection methodologies like the RTL or gate-level, we elaborate on the effective utilization of fault models for increasing the confidence on TSIM-based fault injections. In particular we analyze the effect that applying low-level fault models to the available nets in an RTL processor model causes in architectural registers. Architectural registers are a typical target for performing fault injection using TSIMs [13][25].

In this paper there are 2 main contributions: (i) our extensive injection in the RTL description of a safety-related processor reveals that a single fault activated in the combinational logic or hidden registers propagates into multiple complex errors manifestations in the architectural elements of the processor, and (ii) injection of single error models in higher levels of abstraction is shown to have different effects in terms of dependability than more complex fault models derived from contribution (i).

The analysis and results of this paper show that blindly applying low-level fault models to perform fault injections using TSIM might lead to wrong conclusions. Also, by injecting in a TSIM faults according to the high-level behaviour of faults injected at low-level (e.g., RTL) different error profiles are achieved. In particular, we show that low-level permanent faults manifest as intermittent faults in architectural registers, which may compromise their effective detection.

The rest of this paper is organized as follows. Section II describes the required background on fault models. Section III shows the characterization of the low-level faults model manifestation in architectural registers. In Sections IV and V a case study showing the impact of accurately capturing high-level behavior of low-level fault models is presented and analyzed, respectively. Finally, Section VI presents some related work and our conclusions are drawn in Section VII.

II. Background

The simulation-based robustness verification process requires the definition of suitable fault models accurately representing common cause faults of a given component. In this section we review the most widely known fault models and how these fault models are adapted to deal with several abstraction levels.

A. Low-level Fault Models

In this paper we consider those low-level fault models targeting abstraction levels lower than the microarchitectural level (TSIM) such as gate-level and RTL. Typical examples of low-level fault models are the stuck-at fault model, the open-line, the resistive bridging, and propagation delay. We refer the interested

reader to the fault catalogue given in [28] for a more detailed list of typical microcontroller faults and their corresponding fault-model.

B. Microarchitectural Fault-models

We distinguish two main approaches to perform fault injections using microarchitectural simulators. The first approach consists of using low-level fault models and applying them directly to the possible injection targets. Typical TSIM injection targets are the different memory structures like, RAM and caches [24], and the architectural registers [13], [25]. The second approach consists on defining specific microarchitectural level fault models affecting specific microcontroller modules. For instance, some failures produced in the register data flow are the incorrect selection of input operands or the incorrect access to the register file [4]. For the injection of such complex microarchitectural fault models TSIM must be provided with a detailed definition of the affected structures to be able to model their buggy behavior.

Regardless of the approach used existing TSIM-based injection methodologies are not enough for the derivation of diagnostic coverage and failure rate metrics as required by certification standards such as ISO26262 [9]. However, by increasing the accuracy of TSIM-based fault injection, design flaws can be detected earlier in the design timeline, thus reducing verification time and costs.

III. Characterizing the behavior of RTL fault models

In this section we analyze RTL fault models behavior to understand how injection at the TSIM can be performed providing the maximum possible degree of representativeness.

A. Methodology

We use an RTL description of the Leon3 [27] processor. The Leon3 processor is an in-order processor with integer and floating-point pipelines. Results and conclusions presented in this section while not directly applicable to other processor configurations will likely hold for similar processor architectures but not for significantly different architectures like those of out-of-order processors. From the available nodes of the Leon3 description we inject all the nodes within the integer unit of the processor i.e. all available microcontroller nodes except those representing the cache memories and the register file. The reason for excluding those nodes is twofold. First, caches and register-file data in microcontrollers targeting safety critical applications are typically protected, meaning that faults originated within these structures can be effectively detected. Second, fabs can provide a model for faults directly affecting registers, but for those faults which take place at elements which have a logic path towards them, the only way to account for the impact they have in the former ones is to perform thorough simulation.

Faults have been injected using an injection and analysis tool named FALLES [6] that employs simulation commands as described in [17]. For the faultload we use permanent and transient single fault models. Permanent fault models considered are stuck-at-1, stuck-at-0 and open-line while the transient fault model considered is the transient indetermination [11]. For each permanent fault model, every target node has been injected once at the beginning of the workload, totaling 5246 injection experiments per workload. However, for transient faults 10 injection experiments per target node have been performed, each injection applied at different randomly chosen instants of workload execution. These were active for 1.5 times the clock period, to ensure no time filtering takes place. This allows to improve coverage of the possible outcomes no matter the logic state of the system at the time of injection. The total is 52460 experiments of transient faults per workload.

Several workloads comprising some EEMBC automotive [23] and Mälardalen [12] benchmarks have been executed in the RTL simulator. Results of the golden run (simulation in the absence of faults) are compared with the ones after injection to determine the outcome of fault injections. We have focused on the register file and system registers to determine that an *error* has been produced when a mismatch in the state of these registers is found after comparison with the status of the golden run registers. We focus on the architectural registers only as they are a typical target for injection using TSIM. In the following we present the results of the fault injection experiments at the RTL level.

B. Error multiplicity

In this paper define error multiplicity as the number of different architectural registers a single injected fault can reach and upset during the execution of the workload. Error multiplicity values are shown in Figure 1 for permanent faults and in Figure 2 for transient faults.

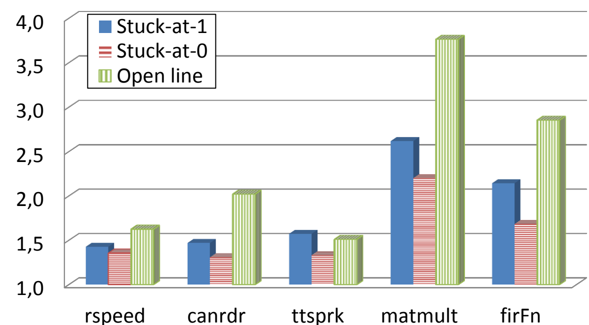

Fig. 1. Errors multiplicity, permanent faults.

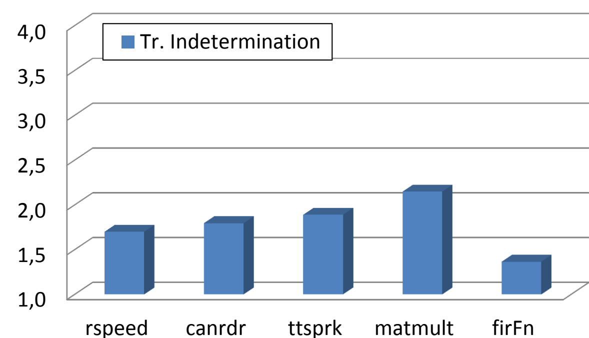

Fig. 2. Errors multiplicity, transient faults.

The figures show a set of interesting information. First of all, in Figure 1 we can appreciate a range which varies between 1.3 and 3.76 affected registers on average, depending on the workload and permanent fault model. What that means is, though executed workload and fault model have an influence, the averaged value can go as high as nearly 4 affected architectural registers per single fault occurred in the integer pipeline. It is not common practice today to test wrong values of 4 different registers concurrently, so the coverage of real-world dependability threats happened at the combinational side

is suboptimal, not considering the extreme cases. Of course, the higher the number of utilized registers by the workload the potentially higher values of multiplicity can appear. It is important to mention that at the RTL open-line faults are propagated as an indeterminate value and thus not logically masked. Therefore, open line multiplicity values instead of representing the actual multiplicity of a physical open line represent an upper bound of the actual error multiplicity.

Moving to Figure 2, a comparison with Figure 1 shows interesting facts. The trend of permanent fault models is not totally followed in this case, where *rspeed, canrdr* and *ttsprk* show rather high values of multiplicity compared to permanent faults, while *matmult* and *firFn* show smaller values. In general, a lower maximum across benchmarks is found for transient faults, what makes sense taking into consideration these faults last much less time active so they should not have time to reach so many registers. Even though, up to 2.14 average multiplicity indicates that a single register injection will not cover adequately the underlying reality.

C. Error repetitions

In order to understand the way a single fault model impacts in the architectural registers throughout time, we perform an analysis on the amount of times an error appears in the same register while executing the benchmark (error repetitions). This information is relevant since an error can 'disappear' when it is overwritten, which masks the error, but then reappear when the logic masking is no longer active. A number of repetitions per benchmark, obtained by averaging the values for every architectural register, is shown for permanent faults in Figure 3 and for transient faults in Figure 4.

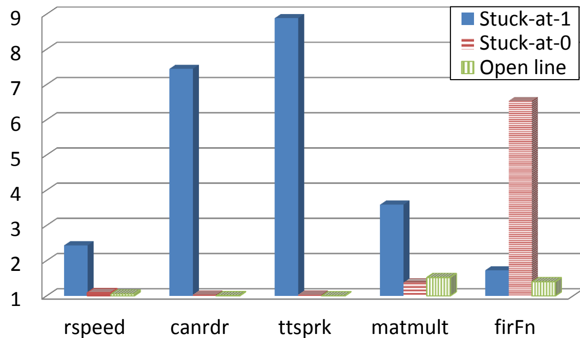

Fig. 3. Errors average repetitions per register, permanent faults.

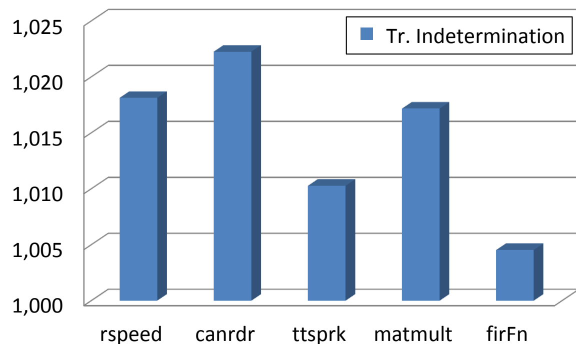

Fig. 4. Errors average repetitions per register, transient faults.

In the first chart, there is a wide disparity across benchmarks and fault models. If we compare across permanent fault models, stuck-at-1 shows the highest number of average repetitions as compared to stuck-at-0 or open-line. This is due to masking affecting more a high logic value than a low one in the case of 4 out of 5 benchmarks, i. e. there are more '1's than '0's in the logic alive values of those benchmarks. Interestingly, in *firFn* the opposite happens, so there is a higher logic masking for '0's. For open-line a reduced value tells that those errors are not easily masked, in the way that when they appear it is strange to see them disappear (but not impossible, since it is not a flat 1 value). This observation is supported with evidence next.

Also, if attention is put on a specific fault model, for instance stuck-at-1, the values change from 1.72 to 8.88 times according to different benchmarks. This means that, contrarily to the intuition of applying a permanent fault model to architectural registers in the higher level assessment, the truth is that an **intermittent** model will better mimic what permanent damage in the combinational logic can cause. To illustrate this fact, in the left hand side of Figure 5, the error duration histogram shows precisely the fact that very few of the errors lasted until the end of execution (i.e. appeared as permanent), as opposed to those which lasted a finite amount of time and were recurrent. If the open line model is studied, the situation is the opposite. In right hand side of Figure 5 the duration histogram of such hardly repeated errors tells that once an open line has hit the architectural registers it is hardly masked since it is not later rewritten with correct values.

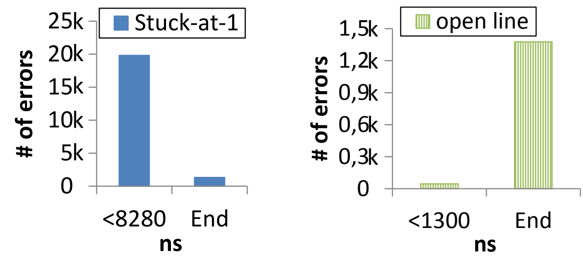

Fig. 5. Durations histogram of Errors in arch. registers for ttsprk, permanent faults.

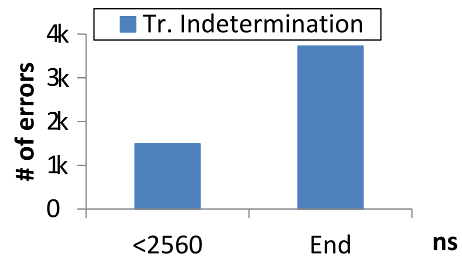

Fig. 6. Durations histogram of Errors in arch. registers for ttsprk, transient faults.

Moving to Figure 4, what is pretty evident is that for transient faults the values are barely above 1, meaning that hardly ever a transient indetermination fault in the pipeline will cause a recurrent error in an architectural register. In other words, when such faults hit the architectural registers they are not masked temporarily but a certain percentage of them is definitely cleared after some time.

D. Error bit upsets

Up to this point, we have acknowledged that multiple and intermittent errors have to be introduced to improve coverage

of what can happen in the on-board embedded systems. In this section we discuss about the error bit upsets i.e, the number of affected bits of each register originated from a single fault.

Figure 7 shows the average number of error bit upsets in architectural registers for permanent fault models, and Figure 8 does the same for transient fault models.

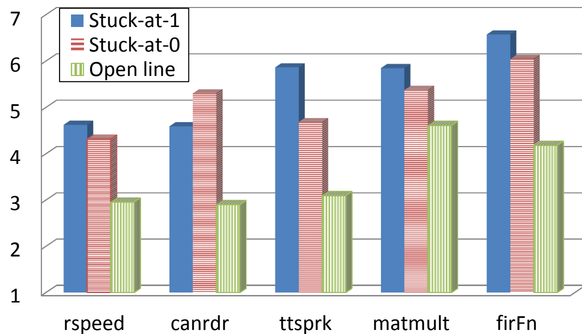

Fig. 7. Average error bit-upsets per register, permanent faults.

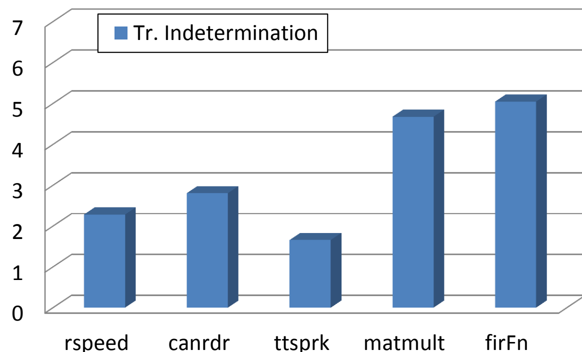

Fig. 8. Average error bit-upsets per register, transient faults.

In the permanent faults case, a minimum of 2.9 and a maximum of 6.57 bits across benchmarks and models (out of 32 bit register sizes) are upset in average. These are rather elevated values, far from single bit upset models.

For the transient faults case, the story is hardly different. It is true though, that a smaller number of bit-upsets applies in general for every benchmark, with special incidence in some of them –see *ttsprk* dropping from 3.09 or more down to 1.65. Only the *ttsprk* case shows average values below 2 considering permanent and transient faults, so again multi-bit models for architectural registers seem to be the safest bet in the representativity context.

IV. Case Study: Modeled Architecture

In this section we analyze how a high-level (TSIM) characterization of low-level (RTL) faults behavior helps increasing the effectiveness of fault injections at high level (TSIM) of abstraction. To do so, we have modeled a lockstep architecture resembling the AURIX [16] processor. Lockstep architectures like the Infineon AURIX [16] and ST microelectronics SPC56EL60L5ST133 [26] processor are complex designs widely employed in the context of highly critical automotive applications for which using effective robustness verification steps using high abstraction level simulators result very beneficial [13], [14].

A. Processor Model

We consider a processor resembling the Infineon AURIX 5-core processor[1] [16]. Such architecture has been modeled with an enhanced version of the SoCLib simulation framework [19] with TriCore binaries [29] to implement a cycle-accurate pipelined in-order core architecture similar to the AURIX processor [16]. We are interested in analyzing the behavior of two of its cores operating in lockstep mode. As in the AURIX processor, those cores operate in a way that the leading thread runs few cycles ahead of the trailing thread. A simple hardware checker is placed in between the trailing core and the shared communication network. During lockstep operation, the hardware checker stalls the bus accesses of the trailing core and snoops leading thread activity (data, interrupts, exceptions, etc.). On each bus access of the leading core, the checker compares the values (address and data if any) against those of the trailing core. On a mismatch an error is reported raising the corresponding interrupt. If no mismatch is detected, the trailing core remains in the same state as the leading core. For instance, leading and trailing cores are allowed to proceed if the memory or I/O request does not require any answer, as for write operations.

B. Applications and System Software

Complex processor architectures like the Infineon AURIX processor are ideal candidates for the consolidation of several applications of different criticalities in the same chip. While in current systems only one highly critical application (e.g., ASIL D) is executed in the platform in the future it is expected having applications of different criticalities integrated onto the same system to maximize the performance provided by recent designs of multicore embedded processors [1].

Currently, with one application running in the lockstep architecture the most effective recovery mechanism consists of resetting and re-starting the execution when an error is detected [15]. However, in a context where several applications can be scheduled to run simultaneously in the same processor this approach is not valid anymore and efficient recovery mechanisms based on effective checkpointing have to be deployed [14]. For this case study we consider several mixed-criticality applications executing simultaneously in the processor and the recovery is based on the effective checkpointing mechanism proposed in [14].

C. Error Injection Methodology

We have performed error injections in the register file using a TSIM of the aforementioned processor. The fault injection has been carried out using two different methodologies: (i) a plain fault injection (PFI) methodology and (ii) a smart fault injection (SFI) methodology. For the PFI we considered low-level fault models cause an analogous low-level effect to the architectural registers. For instance, a transient error is modeled as a single-bit upset in a random bit position of an architectural register whose value can be modified when it is overwritten. We consider transient and permanent fault models.

For the SFI we consider that both transient and permanent faults do not necessarily manifest in architectural registers in the same way they manifest when applied directly to an RTL net. In particular, we use the main properties shown in previous section to build a more complex fault model where multiplicity,

[1]Only 3 cores can be used effectively since the other 2 operate in lockstep mode with 2 of the usable cores.

repetitions, and heterogeneity are taken into account as detailed in the next section.

V. Case Study: Results

For the comparison of PFI and SFI methodologies we ran several EEMBC benchmarks in the simulator and collected statistics. Next, we present the comparison of the approaches focusing on the impact of multiplicity, repetitions, and the multiple bit upsets.

A. Multiplicity

To quantify the impact of error's multiplicity we compare the results of injecting errors using PFI and SFI methodologies. To decouple the rest of effects for this study we consider SFI only takes into account multiplicity effects. In particular, for every fault injection we poison a variable number of registers ranging from 1 to the expected multiplicity value according to the low level fault model considered. We show results only for the case of transient fault models. The analysis for permanent faults is not shown due to lack of space, but conclusions are analogous as for transient faults.

Figure 9 shows the impact of error's multiplicity in the architectural system and user registers. Following the results of the analysis in Section III we consider that the maximum expected multiplicity for the transient indetermination fault model is 2. Results show that even for multiplicity 2 (M2) the error profile changes significantly. On average the percentage of masked faults varies from 39% to 16.6% for the cases of M1 and M2, respectively.

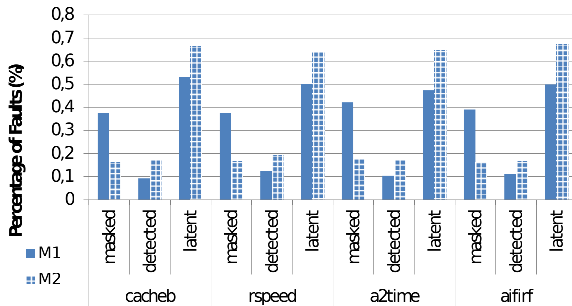

Fig. 9. Effect of multiplicity on error distribution.

B. Error Repetitions

As shown in Section III permanent errors affecting processor internals manifest intermittently in the architectural registers. While the intermittent behavior of permanent errors does not challenge the detection coverage of the lockstep architecture per se, this behavior compromises the effectiveness of error recovery. The reason is that even using a fine-grain checkpointing mechanism like the one proposed in [14] where checkpointing frequency can be increased at low cost, permanent errors might not be effectively (on a timely manner) detected if they are not present in the architectural register at the time the checkpointing mechanism exposes architectural registers to the on-chip bus. Figure 10 shows the percentage of permanent errors that can be detected. Only values for stuck-at-1 (s1) and open-line (ol) fault models are shown in the plot as they represent the two extremes: highly intermittent behavior (s1) and quasi-permanent behavior (ol). For injections we have considered as fault duration the one obtained in the analysis in Section III. As shown in the figure even for the highest checkpointing frequencies in [14] the percentage of faults not effectively detected (per checkpoint) is around 94% for stuck-at-1 but just 3% for open line. When going to lower checkpointing frequencies, those recommended to avoid heavily impacting the performance, the rates for undetected (per checkpoint) stuck-at-1 faults and open line stay the same for the case of an inter-checkpointing interval of 10,000 cycles.

The immediate conclusion we draw from these results is that using a PFI can lead to erroneous conclusions. For example 100% permanent errors detection coverage is expected for permanent fault models when injection is applied directly to the architectural registers [13]. However, as shown, this is far from being the case when faults occur in other components and propagate to the registers since faults can manifest intermittently, hindering detection.

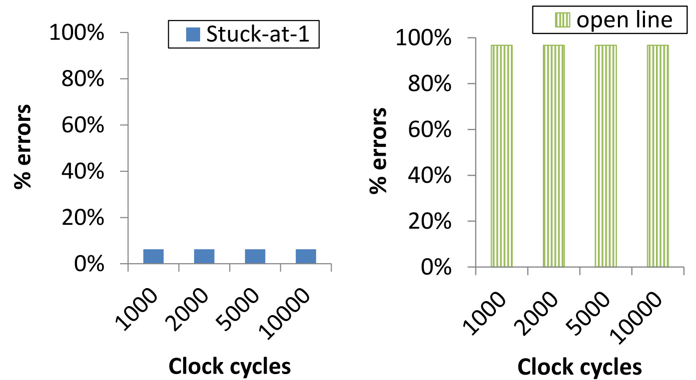

Fig. 10. Percentage of detected errors per checkpoint according to checkpointing period.

C. Error Bit Upsets

To analyze the effects of multiple bit upsets we have carried out fault injection experiments varying the number of bits that are affected in the architectural registers. Figure 11 shows the profile of errors for a single bit affected in the architectural registers (BU1) and when considering two bits (BU2). As shown in the figure the effect of having multiple bit upsets in the architectural registers that are generated within the core is not as important as the effect of multiplicity or error repetitions. For example the masking of faults is decreased only by 7% when considering 2 bit upsets while for the case of a multiplicity of 2 (M2) the masking factor decreased by 2.3×.

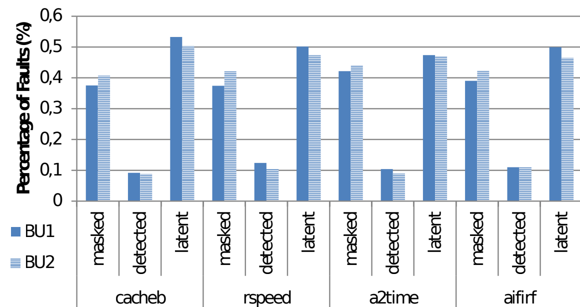

Fig. 11. Effects of multiple bit upsets on error distribution.

VI. Related Work

Fault injection methodologies are widely employed for the microcontrollers robustness verification in the automotive do-

main [22]. Fault injection experiments can be performed at several abstraction levels to exploit the existing accuracy cost trade-off [2]. RTL and gate-level fault injection experiments are the most adopted approaches to perform the certification of hardware products against standards [9].

Practitioners have performed fault injection at the logic and RTL levels using different techniques. A widely-used method is the injection in the HDL through simulator commands [17], which works well for most of the fault models described in the literature.

The majority of works on estimating reliability at the microarchitectural level (TSIM) focus on determining the architectural vulnerability factor (AVF) [21]. The AVF is computed by obtaining the percentage of the architectural bits contributing to the processor's reliability. A similar approach is the one in [3] where the concept of instruction vulnerability factor (IVF) is proposed to evaluate how faults in every instruction affect the final application output.

The suitability of fault models targeting different levels of abstraction was studied in [11] were fault model representativeness was validated for logic/RTL levels. On the contrary, for higher abstraction levels like the TSIM some works have pointed out the difficulties of correlating the results with experiments at the physical level [18] because low-level fault models cannot be directly applied to architectural simulators. A first attempt of correlating TSIM and logic/RTL fault injection experiments was done in [20] focusing on the correspondence between low-level fault models and microarchitectural failure scenarios. Later, authors in [7] showed that RTL fault injection experiments can be correlated with architectural metrics for the case of permanent faults. However, up to our knowledge, none of the previous studies analyzed the main manifestation properties of RTL fault models in the architectural registers for the presented fault models and how high-level fault models can be derived effectively to perform TSIM-based fault injection.

VII. Conclusions

Microcontroller verification based on fault-injection is a key approach to carry out the robustness verification step of safety-critical systems. TSIM fault injection has been regarded recently as a low-cost, flexible and accurate framework platform for fault injection. However, so far fault models have been applied in the registers of a TSIM in the same way as in the nodes of RTL or gate-level descriptions.

In this paper we show that the behavior of faults in the registers – the main visible component in a TSIM – differs significantly from the original fault models injected in combinational nodes. Therefore, our analysis proves that fault models in low-level descriptions need to be applied differently in the registers of a TSIM. We support this conclusion by comparing the error detection capabilities of an automotive microcontroller in front of the original fault models applied to registers and their true manifestation in the registers obtained from RTL fault injection.

Acknowledgments

The research leading to these results has received funding from the Ministry of Science and Technology of Spain under contract TIN2015-65316-P and the HiPEAC Network of Excellence. Carles Hernández is jointly funded by the Spanish Ministry of Economy and Competitiveness (MINECO) and FEDER funds through grant TIN2014-60404-JIN. Jaume Abella has been partially supported by the MINECO under Ramon y Cajal postdoctoral fellowship number RYC-2013-14717.

References

[1] Nvidia tegra k1 embedded platform design guide, 2015. https://developer.nvidia.com/.
[2] ARTEMIS Joint Undertaking. *VeTeSS project: www.vetess.eu*.
[3] D. Borodin and B. H. Juurlink. Protective redundancy overhead reduction using instruction vulnerability factor. In *CF*, 2010.
[4] J. Carretero, P. Chaparro, X. Vera, J. Abella, and A. González. End-to-end register data-flow continuous self-test. In *ISCA*, 2009.
[5] I. E. Commission. *IEC 61508 Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems*.
[6] J. Espinosa, D. de Andres, J. Ruiz, C. Hernandez, and J. Abella. Towards certification-aware fault injection methodologies using virtual prototypes. In *FDL WiP*, 2015.
[7] J. Espinosa, C. Hernandez, J. Abella, D. de Andres, and J. C. Ruiz. Analysis and rtl correlation of instruction set simulators for automotive microcontroller robustness verification. In *DAC*, 2015.
[8] ETAS. Solutions for embedded testing, 2014. http://www.etas.com/data/presentations/SolutionsforEmbeddedTesting.pdf.
[9] I. O. for Standardization. *ISO/DIS 26262. Road Vehicles – Functional Safety*, 2009.
[10] E. Francis. Autonomous cars: no longer just science fiction. *Automotive Industries*, 193, 2014.
[11] P. Gil, J. Arlat, H. Madeira, Y. Crouzet, T. Jarboui, K. Kanoun, T. Marteau, J. Dures, M. Vieira, D. Gil, J. C. Baraza, and J. Gracia. Fault representativeness. Technical report, DBench project, IST 2000-25425 [Online]. Available: http://www.laas.fr/DBench, 2002.
[12] J. Gustafsson, A. Betts, A. Ermedahl, and B. Lisper. The Mälardalen WCET benchmarks – past, present and future. In *WCET Workshop*, 2010.
[13] C. Hernandez and J. Abella. Live: Timely error detection in light-lockstep safety critical systems. In *DAC*, 2014.
[14] C. Hernández and J. Abella. Low-cost checkpointing in automotive safety-relevant systems. In *DATE*, 2015.
[15] C. Hernández and J. Abella. Timely error detection for effective recovery in light-lockstep automotive systems. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 34(11):1718–1729, 2015.
[16] Infineon. AURIX - TriCore datasheet. highly integrated and performance optimized 32-bit microcontrollers for automotive and industrial applications, 2012. https://www.infineon.com/dgdl?folderId=db3a304412b407950112b409ae660342&fileId=db3a30431f848401011fc664882a7648.
[17] E. Jenn, J. Arlat, M. Rimen, J. Ohlsson, and J. Karlsson. Fault injection into VHDL models: the mefisto tool. In *FTCS*, 1994.
[18] M.-L. Li, P. Ramachandran, U. Karpuzcu, S. Hari, and S. Adve. Accurate microarchitecture-level fault modeling for studying hardware faults. In *HPCA*, 2009.
[19] LiP6. Soclib, 2003-2012. http://www.soclib.fr/trac/dev.
[20] M. Maniatakos, N. Karimi, C. Tirumurti, A. Jas, and Y. Makris. Instruction-level impact analysis of low-level faults in a modern microprocessor controller. *Computers, IEEE Transactions on*, 60(9):1260–1273, Sept 2011.
[21] S. Mukherjee, C. Weaver, J. Emer, S. Reinhardt, and T. Austin. A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. In *MICRO*, 2003.
[22] J.-H. Oetjens et al. Safety evaluation of automotive electronics using virtual prototypes: State of the art and research challenges. In *DAC*, 2014.
[23] J. Poovey. *Characterization of the EEMBC Benchmark Suite*. North Carolina State University, 2007.
[24] D. Sánchez, Y. Sazeides, J. M. Cebrián, J. M. García, and J. L. Aragón. Modeling the impact of permanent faults in caches. *ACM Trans. Archit. Code Optim.*, 10(4):29:1–29:23, dec 2013.
[25] B. Sangchoolie, F. Ayatolahi, R. Johansson, and J. Karlsson. A study of the impact of bit-flip errors on programs compiled with different optimization levels. In *EDCC*, 2014.
[26] STMicroelectronics. *32-bit Power Architecture microcontroller for automotive SIL3/ASILD chassis and safety applications*, 2014.
[27] http://www.gaisler.com/cms/index.php?option=com_content&task=view&id=13&Itemid=53. *Leon3 Processor*. Aeroflex Gaisler.
[28] VeTeSS project: www.vetess.eu. *D5.2: Analysis of Fault Types and Common-Cause Faults*.
[29] J. Wetzel, E. Silha, C. May, B. Frey, J. Furukawa, and G. Frazier. *PowerPC User Instruction Set Architecture*. IBM Corporation, 2005.