Randomization for Safer, more Reliable and Secure, High-Performance Automotive Processors

David Trilla[†,‡], Carles Hernandez[†], Jaume Abella[†], Francisco J. Cazorla[⋆,†]
[†] Barcelona Supercomputing Center (BSC). Barcelona, Spain
[‡] Universitat Politècnica de Catalunya (UPC), Barcelona, Spain
[⋆] Spanish National Research Council (IIIA-CSIC). Barcelona, Spain.

***Abstract*—The automotive domain is witnessing a relentless transition to autonomous cars demanding high-performance processors to timely execute complex, critical, decision-making software. The other side of the coin is that high-performance processors include hardware features like shared multilevel caches and multiple cores that expose the system to significant security threats, challenge time predictability, and jeopardize reliable operation due to the use of advanced process technology. In this paper, we discuss how introducing randomization in the non-functional behavior of certain hardware components helps to achieve a three-fold objective while preserving high-average performance capabilities of high-performance processors: improving the security of complex processors, favoring time predictability via probabilistic analysis, and enhancing reliability against aging and voltage noise.**

***Index Terms*—Real-Time Embedded Systems, Security, Reliability, Time-Predictability**

I. Introduction

The design of processors for the automotive domain has been traditionally driven by safety constraints. This is defined in domain-specific standards, e.g. ISO26262, that specify for each automotive safety integrity level (ASIL) the maximum failure rate that is tolerated as well as the safety requirements.

The advent of autonomous driving cars is forcing automotive industry to adopt massively-parallel processors delivering much higher performance to timely execute complex data-management and decision-making software, which already comprises more than 100 million lines of code [12]. Recently, some high-performance processor designs have been proposed for the automotive domain such as those in the NVIDIA DrivePX and the RENESAS R-Car H3 platforms. Unfortunately, high-performance processors do not only bring higher computational capabilities, but an associated unprecedented complexity that creates difficulties in the verification of the security, reliability, and time predictability properties of the system. To make things worse, despite some synergies, the typical countermeasures to handle complexity for each of these metrics are to some extent mutually exclusive. For instance verifying reliability properties requires detailed information about processor internals (observability) while security demands for the opposite (opacity). Likewise, hardware features like caches help to improve the average performance of the system but challenge predictability and security due to the existing dependence between input data and execution time.

Interestingly, injecting randomization at different layers of the computing system has been shown beneficial in different domains to optimize metrics such as security, resilience to software bugs, hardware reliability and even timing analysis. As an illustrative example, for security, randomization techniques reduce the possibility of inferring information from experimentation since execution time does not correlate any longer with the particular values of sensitive data, thus challenging malicious attempts to obtain information. For the case of cache side-channel attacks, memory layout randomization avoids inter-task conflicts in the cache between the victim and the attacker processes, preventing these conflicts from leaking information since the memory layout, and so cache conflicts, is different and random for every re-execution.

In this paper, with focus on the future computing platforms that will be deployed in the automotive domain, we make the following contributions:

1) From a conceptual point of view, we show how injecting randomization in hardware/software non-functional behavior can together protect against certain security threats, improve hardware reliability, and time predictability of complex processors in automotive, with minimum effects on average performance.
2) From a practical point of view, we discuss how randomization techniques align with the automotive industry requirements. We also illustrate the associated average performance and implementation costs of randomization by respectively analyzing the performance and modifications applied to a commercially available multicore processor targeting the space domain.

Overall, we show that injecting randomization in future high-performance processor designs for cars (Section II) is a promising solution to cover automotive security (section III), time predictability (Section IV), and reliability (Section V) needs of future automotive chips while its implementation and average-performance overheads can be contained (section VI).

II. Hardware and Software Evolution in Automotive Computing Systems

The demand for higher levels of automation carries huge changes on the software side. Software implements critical functionalities such as learning and adaptation to the environment, and manages huge and diverse sets of data. In fact, software is becoming one of the main competitive factors in every car generation. This has resulted in the software footprint in cars already reaching hundreds of millions of lines of code, and an increase in software's performance needs. According to ARM prospects achieving fully autonomous cars requires

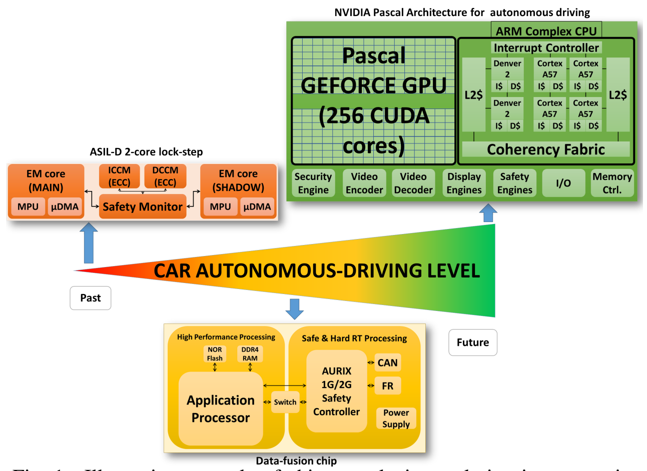


Fig. 1. Illustrative example of chip complexity evolution in automotive.

an unprecedented increase in computing power, up to 100X with respect to that achievable in 2016.

At hardware level, the integration of new functionalities in cars forces manufacturers to include not only a higher number of on-board electronic control units (ECU), but also replacing some of these relatively simple microcontrollers by much more complex processor designs as a way to contain their number. However, one of the most critical aspects of using high-performance processor designs in the automotive domain relates to the increase of the complexity and cost of the verification process that aims at showing that designs comply with the robustness criteria specified in the ISO26262 standard. Figure 1 shows an illustrative architecture of today's lock-step ECU devoted to ASIL-D applications, a complex processor used for driving assistance systems, and a computing platform proposed for autonomous cars. As shown, the degree of hardware complexity and parallelism – reaching its maximum exponent with Graphics Processing Units (GPUs) – increases significantly to respond to the exponential increase in performance demands. In lock-step CPUs, critical applications are relatively simple and low-performance demanding. In the data-fusion chip, critical applications require some more performance – provided by the ASIL-D AURIX processor – while high-performance applications execute on a lower ASIL (e.g. ASIL-B) hardware. Finally, in the chip for self-driving we do not see the separation among the safety processing and the high-performance processing. This is due to the convergence between both to effectively provide autonomous driving.

Unfortunately, the necessary inclusion of high-performance processors in cars brings new challenges not faced so far in the automotive domain. First, high-performance features included in these processors complicate the verification process of both timing and functional correctness. For time predictability the inclusion of a higher number of stateful resources significantly complicates deriving trustworthy execution time bounds. Second, smaller and inevitably more vulnerable technology nodes are required to accommodate a higher amount of hardware resources in a single chip which clashes with the robustness needs imposed by ISO26262. And third, the new hardware features (e.g. caches and shared resources) produce software execution time variability and interferences between different software components, that can be used by external attackers to extract sensitive information such as cryptographic keys through side-channel attacks.

Complex computing hardware prevents managing automotive security, time-predictability and reliability with per-component ad-hoc solutions requiring specific verification means given that verification cost is already huge in much simpler platforms. Instead, a global strategy is needed to address security, time-predictability and reliability holistically without significantly increasing verification costs.

III. Security

Connected vehicles have become one of the major goals of car makers given their potential to, for instance, provide the user with new software updates that add new features and enhance existing functionality. These interconnection capabilities compounded with the utilization of high-performance processor, can be used by malicious software to perform several types of attacks. In particular, we focus on: side channel attacks, unauthorized control information tampering and denial of service.

A. *Unauthorized control information tampering (UCIT)*

UCIT vulnerabilities include many common software-related security problems such as buffer overflow, format string, integer overflow, and double-freeing of heap buffer. Attackers exploit these vulnerabilities by changing control information with the purpose of pointing to the attacker's malicious code. These attacks do not depend on the potential actions on the user side but simply exploit existing program bugs to attack the system.

Benefits of Randomization. Randomization offers a path to address UCIT attacks by relocating both position independent and dependent regions either by software or hardware means. Different randomization schemes based on memory layout randomization can effectively protect the system from UCIT vulnerabilities by randomizing the locations of critical data elements and thus, making difficult, if at all possible, for an attacker to exactly determine the runtime locations of vulnerable points using experimentation. Coarse-grain randomization mechanism like the transparent runtime randomization [5] suffice to protect the system from UCIT vulnerabilities. While they fail to meet other properties like protection against SCA, fine-grain randomization mechanisms like [1] can provide that protection.

B. *Denial of Service (DoS)*

In high-performance multicore processors, some key resources are shared among running tasks. Resource sharing allows processors to improve area and power efficiency but introduce a side effect on the security, and also time-analyzability, when processors do not provide sufficient performance isolation properties. Multicore processors are vulnerable to DoS attacks since one shared resource, typically the memory system, can be unfairly shared amongst multiple cores. In this context, a malicious task can compromise the performance of another task running in the same processor by clogging a shared resource, significantly affecting the performance of co-running tasks or even precluding resource utilization by others. Intuitively, one may think that this effect only arises in processor designs for the mainstream market with limited performance isolation properties, however, this effect has also been observed in some processors targeting the real-time domain [8].

DoS attacks can be prevented by allowing a fair utilization of shared resources. This requires (1) a proper dimensioning of system resources with per-core dedicated buffers to avoid scenarios where an attacker stalls a shared resource, and (2) adequate arbitration policies.

Benefits of Randomization. A time-randomized processor meeting these goals is presented in [11]. It ensures a fair utilization of shared resources by (1) limiting the number of in-flight requests for each core (thus limiting the maximum delay a request can suffer due to contention), and (2) implementing a random arbitration policy that accounts for the time each core uses shared resources [11]. Hence, shared resource bandwidth is fairly shared across cores regardless of their timing behavior. Moreover, such time-randomized arbitration policy is compatible with AMBA protocols for communications, since AMBA adds no constraints on the particular arbitration policy used.

C. Side-channel attacks (SCA)

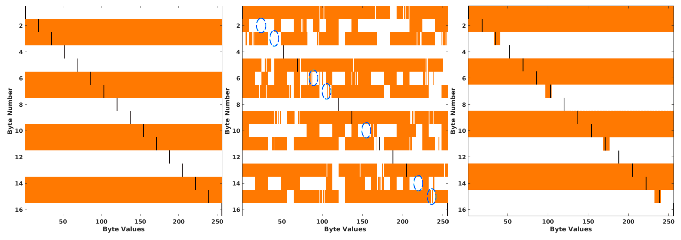
SCA extract secret key data by exploiting the information leakage resulting from the physical implementation of the system. An important bulk of SCA exploits the information leakage of the hardware through input data dependent execution time. The most common timing-related SCA exploit cache behavior to obtain sensitive information. There are several main types of SCA able to exploit different cache properties.

Contention-based attacks. In this case the, attacker contend for the same cache set with the victim process, potentially leading to eviction of one's cache line by the other. When the contention and eviction is deterministic, the attacker can infer the memory address (determined by the value accessed) of the victim based on the cache sets that have been accessed.

Benefits of Randomization. Layout randomization [11], [10] is an effective mechanism to protect against contention-based attacks [10]. This is so because, when layout randomization is applied, conflicts in the cache cannot be deterministically mapped to specific addresses and thus, there is no information flow between the victim and the attacker.

Reuse-based attacks. These attacks exploit shorter execution times experienced by memory accesses when fetched data blocks are stored in the cache (i.e they are reused).

Benefits of Randomization. Unfortunately, layout randomization is not enough to protect against reuse-based attacks since this sort of attacks exploit the fact that data that is already in the cache will be accessed faster regardless of the location of cache memory lines. Interestingly, the only existing mechanism able to protect against reuse-based attacks that allows using caches is also based on randomization [2]. This mechanism implements a randomized fetching algorithm that decouples the presence of data in cache from the actual request of data, thus confusing the attacker when assessing if blocks already present in the cache are being requested.



(a) Deterministic (b) Randomized (c) Secure

Fig. 2. Side-channel attack key value candidates. White cells show discarded value; orange cells possible candidates for the key; and black cells the actual key value identified by the attacker.

Power attacks. The amount of power dissipated by a program can also leak cryptographic information. When instructions execute fixed-time repetitive operations, like cryptographic algorithms that use multiple iterations for a given secret key, attackers can match power profiles obtained to infer the cryptographic data.

Benefits of Randomization. Randomizing the execution time to achieve protection against power analysis attacks was proposed in [3] by introducing random noise via randomly interleaving dummy instructions with the actual code when the execution of encryption algorithms is detected. However, memory layout randomization schemes such as those implemented in [11] already randomize the execution time exhibited by the processor thus being a better option to protect from both sources of attacks, namely contention-based and power analysis attacks.

D. Illustrative SCA Example

Figure 2 shows the results of a timing SCA trying to recover the secret key being used for AES encryption. In particular, the experiment performs Bernstein's AES SCA in which an intruder measures the execution time of encryption rounds on a victim's machine and a copy of the same hardware and software, and then correlates the timing results to extract the value of the AES secret key (16 Bytes). This attack relies on cache conflicts to reveal information. We performed the attack on a cycle accurate simulator and compared to which extent the attacker is able to narrow down the correct value of the secret key in a deterministic cache microarchitecture against a non-deterministic one.

Figure 2(a) shows some results for a non-randomized (deterministic) architecture. Black marks show the secret key values belonging to candidate values (in orange). We observe how some bytes are obtained (e.g. byte 1) and others require brute force to be retrieved (e.g. byte 2). In the case of the randomized design [11] (Figure 2(b)) the attack discards the correct value for some bytes of the key (blue circles). Hence, a brute force attack on the values regarded as candidates will be incapable of finding the key. For the sake of completeness, we also performed the attack on a processor with caches implementing a solution against specific attacks [10] on Figure 2(c). As it can be observed, the attacker is still able to retrieve most of the values of the secret key. This is due to the fact that the particular implementation of a secure cache fails to isolate the leakage of information under the presence of internal interference, which is the vector of attack for Bernstein's method.

IV. Time Predictability

Automotive systems must undergo an assessment of their functional and non-functional properties. For the later ISO26262 part 6 includes the requirement that in the software verification step an estimation of the upper bound of execution time of programs shall be made. This upper-bound is usually known as Worst-Case Execution Time (WCET). To do so, real-time systems employed in the automotive domain have relied so far on using simple processor (predictable) so an upper-bound on the execution time of a task can be estimated with affordable verification costs.

With current industrial practice, the quality a WCET estimate, predominantly builds on previous experience and engineering judgment. A common industrial practice is Measurement-Based Timing Analysis (MBTA) that consists in running a set of tests aimed at triggering the worst possible situations, measuring execution time and recording the maximum observed value, or High-Water Mark (HWM). A safety margin (e.g. 20% in single-core processors) is added to the HWM, to compensate for the uncertainties of the testing process. Parallel high-performance hardware complicates deriving (and providing evidence of the correctness of) the margin. With MBTA, guaranteeing that tests hit the worst-case (or a good approximation to it) becomes infeasible as hardware complexity grows: analyzing the worst-possible execution conditions requires precise control of low-level architectural features with significant impact on execution time variation.

Benefits of Randomization. Removing jitter from hardware components by forcing them to work on their worst latency eases timing analyzability while increasing the security of the system. However, this may cause large performance losses, thus limiting the computational benefits of multicore processors. This approach has been the one traditionally followed in low-performance time-critical systems, which do not match the performance expectations of future automotive applications. Injecting randomization in the hardware resources exhibiting jitter simplifies the application of statistical techniques to derive the worst-case execution time (WCET) for the software [11]. This, in turn, simplifies reasoning on the probability of different execution times to occur, so that the value whose exceedance probability can be deemed as low enough according to safety standards (i.e. ISO26262), can therefore be used as WCET estimation.

A. LEOPARD: a time-randomized multicore processor

As an illustrative example of the feasibility of the use of randomization, we describe LEOPARD, an enhanced 4-core LEON3 architecture [11]. LEOPARD implements randomization in cache placement/replacement and arbitration policies,

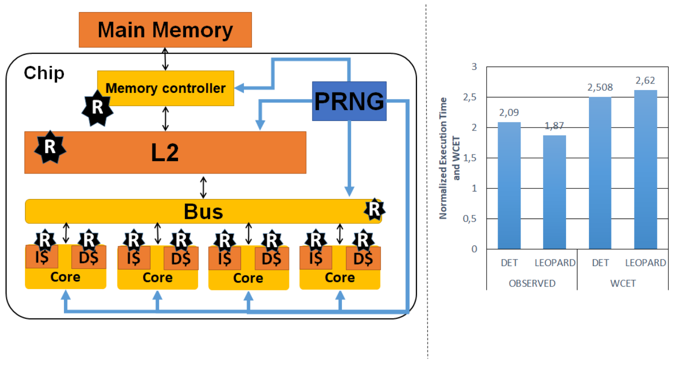


Fig. 3. Leopard block diagram and WCET results for a Space application [11]. Processor blocks including randomization features are marked with a $\star$. Note that $I\$$ and $D\$$ respectively stand for instruction and data cache, while PRNG stands for pseudo-random number generator (see Section VI-B)

as shown in Figure 3. A low overhead pseudo-random number generator (PRNG) produces 32-bit sequences of bits which are divided into small sequences and sent to the different randomized resources. The PRNG incurs low-overhead and produces long-enough sequences to prevent correlation among randomized events [11]. By using random placement policies, LEOPARD defeats SCA attacks, which cannot infer addresses – and hence data – from the impact of cache conflicts in timing behavior. Also, since execution time – and hence energy consumption – is randomized to some degree, LEOPARD challenges power analysis attacks. LEOPARD limits per-core in-flight requests in shared resources and implements a random credit-based arbitration for those resources, thus avoiding an attacker to hog shared resources, as needed to prevent DoS attacks. This design, by providing randomized timing behavior and appropriate random placement policies, also enables reliable operation since it meets the requirements described in Section V. Figure 3 (right) shows observed performance and WCET estimates obtained with a critical real-time control function running together with 3 memory intensive payload applications on LEOPARD and on its time-deterministic (DET) counterpart. Note that the WCET for DET is less trustworthy since an arbitrary 20% margin on top of the highest observed execution time is used. Instead, probabilistic tools are used on top of LEOPARD measurements, which produce quantifiable confidence [11]. All results are normalized w.r.t. single-core execution with no contention. For both, observed execution time (average performance) and WCET estimates, randomization provides competitive results with higher confidence on obtained WCET bounds.

V. Reliability

One key metric in the design of electronic automotive components is its life expectancy, which for current cars is settled around 10 years. On the contrary, the lifetime of high-performance processor is much shorter ($\approx$ 5 years). While the utilization of more robust technology nodes and reduced operating frequencies will extend high-performance processors lifetime, new architectural solutions are also required to contribute to reaching this goal and ensure lower failure rates during the lifetime of the devices. High-performance hardware features impact non-functional aspects other than performance and may create systematic repetitive patterns either in time or in space. For instance, conventional cache memories build upon modulo placement so that location of data is determined by its address. Hence, this may make some cache regions to be highly utilized whereas others remain unused. Sources of aging such as Hot-Carrier Injection, whose impact is directly proportional to utilization, can therefore lead to early failures if cache space is used very heterogeneously. Alternatively, patterns can occur also in the time dimension if events such as, for instance, memory accesses, occur with precise frequencies. Using deterministic caches, and arbiters in interconnects and memory controllers, can create those systematic patterns. A side effect is that power dissipation follows those patterns. The synchronization of power demanding events and the frequency at which those events occur has been shown to be the factors with major contribution to the voltage noise in the power distribution networks of multicore processors [9]. Voltage noise is created by power fluctuations and this effect is amplified when such fluctuations are repetitively caused by the synchronization of high power consuming events. This may cause severe voltage droops and hence, failures affecting all tasks running in the processor.

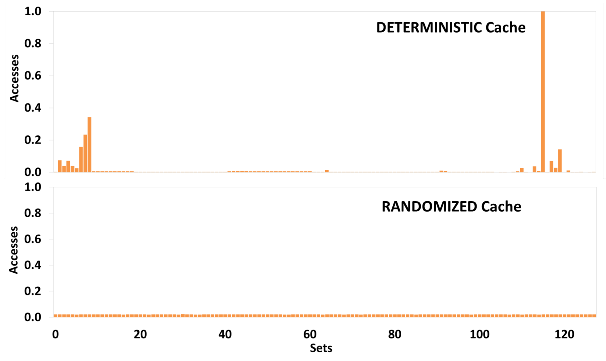
Benefits of Randomization. We illustrate how randomization can help increasing reliability with two examples.

A. Illustrative Example I

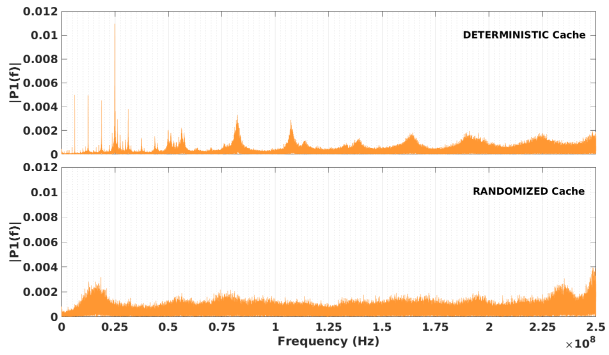
In caches conventional placement algorithms such as modulo, access distribution across sets is completely address dependent. Instead, when cache placement is randomized [11], accesses to the cache sets are randomly distributed since random placement algorithms employ a combination of address tags bits with random bits of a seed to generate the index cache contents so every time the random seed is modified a different set is accessed for any given address. Having a highly biased cache set utilization is expected to lead to higher degradation due to, for instance, hot carrier injection, whose effects are proportional to utilization as indicated before. This is illustrated in Figure 4(a), where we show the normalized access distribution to the sets of a well-known automotive benchmark to a level 1 cache (128 sets). We observe how modulo placement (deterministic) can easily result in a concentration of accesses in few sets (top plot), thus leading to early degradation. Instead, random placement (bottom plot) guarantees a homogeneous distribution of sets over time, thus minimizing maximum degradation. Note that utilization in both plots is normalized w.r.t. the highest utilization of any set in the non-randomized design.

B. Illustrative Example II

In conventional processor architectures pathological scenarios can lead to the systematic occurrence of recurrent high power demanding events. In processors including randomization the occurrence likelihood of extreme situations is not only reduced but also it can be measured quantitatively using statistical tools, which allows to properly dimension the system [9]. Figure 4(b) shows the frequency spectrum resulting from the application of the Fast Fourier Transform to the power profile of a pathological benchmark executed in a conventional processor (top) and in a time-randomized one (bottom). As shown in the plot, in a time-randomized processor the synchronization of power demanding events is much less significant leading to an almost flat spectral behavior. The power measurements have been sampled from McPAT, a power simulator, that models power dissipation through the use of performance monitoring counters. McPAT has been set to model an automotive processor at 0.9V and 22nm technology.



(a) Normalized access distribution to level 1 data cache sets for `a2time` for modulo (top) and randomized placement (bottom)



(b) Frequency spectrum of power dissipation for a regular processor (top) and a time randomized processor (bottom)

Fig. 4. Power dissipation and access distribution across sets

VI. Randomization for Automotive Systems

The utilization of randomization techniques in the context of automotive systems may bring some difficulties. In general, safety critical systems have traditionally advocated for hardware computing platforms in which the timing behaviour of applications is as deterministic as possible to ease the timing verification. However, the current game-changing situation caused by the advent of complex high-performance hardware clashes with classical real-time deterministic platforms.

The contribution of this paper aims at building the proper argument over a holistic randomization approach as the one proposed in [11] in which randomization does not only bring benefits to security but also to reliability and predictability.

When randomization is applied in this manner the timing behaviour of the processor can be easily modeled with probabilistic timing analysis techniques. Therefore, in our view, the adoption of randomization-based solutions (on which we elaborate next) although not trivial and might require some changes to standards, it is not going to find a roadblock due to certification.

A. Probabilistic Reasoning in Certification and Qualification

Security. The Common Criteria (CC) standard for security certification does not strictly impose any specific method for certification, nor imposes the certification of the system itself. Still, CC is widely used to certify security aspects of systems in different domains. Hence, CC is agnostic on whether the system uses or not randomization for security purposes. In fact, probabilistic and statistical reasoning built upon randomization is a very convenient fit for Evaluation Assurance Levels (EAL), which describe the degree of rigor and depth of the evaluation of the security aspects (from 1 to 7, being EAL-7 the highest). In particular, there is an excellent match between the quantitative evaluation of randomization for security and the evidence needed to reach a specific EAL.

Timing Analysis. Authors in [4] have shown how to fit execution time exceedance into a residual random fault in ISO26262. Using such approach timing verification engineers can attack the predictability challenges associated with the utilization of complex hardware in an affordable and systematic manner. In this context, the utilization randomization techniques are a key element to ease the process of quantifying the residual risk associated with the timing violations of software functions.

Reliability. Random hardware faults are already part of ISO26262, which provides guidelines regarding coverage and evidence required for different ASIL levels. Therefore, addressing reliability considerations by means of probabilistic reasoning built upon randomization is a perfect fit for certification against ISO26262.

B. Overheads of randomization

Randomization of non-functional aspects such as cache placement or arbitration policies of shared resources can be achieved with hardware and software means. Taking as an example the cache, cache layout randomization can be implemented at software level by dynamically or statically reallocating code, heap and stack at runtime [1], [5], [6]. The latter solution, that is based on generating different versions of the source code each with a random order in the definition of functions, a random padding among them, a randomization of the locals and globals, has been shown compatible with ISO26262 [6]. Hardware randomization, implemented in caches and arbiters of shared resources, requires hardware modifications that impact aspects such as power, area and performance. These overheads have been analyzed for the LEOPARD processor, showing them to be very low (typically below 2% for EEMBC and Mediabench workloads [7]). Hence, we regard those overheads as low to make randomization an attractive approach to deal with security, predictability and reliability concerns. Randomization has also been implemented in the shared L2 caches in LEOPARD. Including randomization in shared caches does not impose additional hardware overheads compared to a private cache. The main implications come from the need for handling seed across different tasks. Additionally, including randomization in shared caches also allows to deploy mechanisms to handle inter-task interferences beyond simple partitioning schemes since in randomized caches inter-task evictions depend only on the frequency at which contending task are able to evict data from the shared cache. In general, randomization techniques provide more efficient ways to handle interferences in processors with intensive resource sharing.

VII. Conclusions

Driven by the unprecedented demand for high-performance, automotive chip providers are embracing aggressive parallel hardware designs as the only feasible approach to cover those demands. However, high-performance hardware has side effects on validation and verification of security, reliability and predictability requirements. We have analyzed the opportunities brought by randomization to provide a feasible path to address these challenges. We have further analyzed randomization overheads and how it fits, or require minimum changes to fit, security (Common Criteria) and safety (ISO26262) standards in the automotive domain.

Acknowledgments

This work has been partially funded by the Spanish Ministry of Science and Innovation under grant TIN2015-65316-P. Jaume Abella has been partially supported by the Ministry of Economy and Competitiveness under Ramon y Cajal fellowship number RYC-2013-14717.

References

[1] C. Curtsinger et al. Stabilizer: Statistically sound performance evaluation. *SIGARCH Comput. Archit. News*, 41(1):219–228, March 2013.

[2] F. Liu et al. Random fill cache architecture. In *MICRO*, pages 203–215, Dec 2014.

[3] H. Qu et al. A random delay design of processor against power analysis attacks. In *ICSICT*, pages 254–256, 2010.

[4] I. Agirre et al. Fitting software execution-time exceedance into a residual random fault in iso-26262. *IEEE Transactions on Reliability*, pages 1–14, 2018.

[5] J. Xu et al. Transparent runtime randomization for security. In *SRDS*, pages 260–269, Oct 2003.

[6] L. Kosmidis et al. Containing timing-related certification cost in automotive systems deploying complex hardware. In *DAC*, pages 22:1–22:6, New York, NY, USA, 2014. ACM.

[7] P. Benedicte et al. Design and integration of hierarchical-placement multi-level caches for real-time systems. In *DATE*, 2018.

[8] P. K. Valsan et al. Taming non-blocking caches to improve isolation in multicore real-time systems. In *RTAS*, 2016.

[9] R. Bertran et al. Voltage noise in multi-core processors: Empirical characterization and optimization opportunities. In *MICRO*, pages 368–380, 2014.

[10] Z. Wang et al. A novel cache architecture with enhanced performance and security. In *MICRO*, pages 83–93, Nov 2008.

[11] C. Hernández. Design and implementation of a time predictable processor: Evaluation with a space case study. In *ECRTS*, pages 16:1–16:23, 2017.

[12] J. Owens. Delphi automotive, the design of innovation that drives tomorrow. Keynote talk. In *DAC*, July 2015.

David Trilla is a PhD. Student for the CAOS group at BSC. He obtained his M.S. degree in 2016 fromt the Universitat Politecnica de Catalunya. He enrolled BSC in 2014 and his current research focuses on the effects on energy consumption, security and reliability on randomized architectures. **Contact Information:** Nexus II Building. c/ Jordi Girona, 29. 08034 Barcelona (Spain). phone: +34 934137166. mail: david.trilla@bsc.es.

Jaume Abella is a senior PhD. Researcher at BSC and HiPEAC member. He received his MS (2002) and PhD. (2005) degrees from the UPC. He worked at the Intel Barcelona Research Center (2005-2009) in microarchitectures for fault-tolerance and low power, and memory hierarchies. He joined the BSC in 2009 where he is in charge of hardware designs for FP7 PROXIMA, and BSC tasks in H2020 SAFURE. **Contact Information:** Nexus II Building. c/ Jordi Girona, 29. 08034 Barcelona (Spain). phone: +34 934137609. mail: jaume.abella@bsc.es.

Carles Hernandez received the M.S. degree in telecommunications and PhD in computer sciences from Universitat Politecnica de Valencia, in 2006 and 2012, respectively. He is currently senior PhD. Researcher at the Barcelona Supercomputing Center. His area of expertise includes network-on chip and reliability-aware processor design. He participates (has participated) in NaNoC, parMERASA, PROXIMA IP7 and VeTeSS ARTEMIS projects. **Contact Information:** Nexus II Building. c/ Jordi Girona, 29. 08034 Barcelona (Spain). phone: +34 934137170. mail: carles.hernandez@bsc.es.

Francisco J. Cazorla is the leader of the CAOS group at BSC and member of HIPEAC Network of Excellence. He has led projects funded by industry (IBM and Sun Microsystems), by the European Space Agency (ESA) and public-funded projects (FP7 PROARTIS project and FP7 PROXIMA project). He has participated in FP6 (SARC) and FP7 Projects (MERASA, VeTeSS, parMERASA). His research area focuses on multithreaded for both high-performance and real-time systems on which he is co-advising several PhD theses. **Contact Information:** Nexus II Building. c/ Jordi Girona, 29. 08034 Barcelona (Spain). phone: +34 934137173. mail: francisco.cazrola@bsc.es.