Security, Reliability and Test Aspects of the RISC-V Ecosystem

Jaume Abella [1], Sergi Alcaide [1], Jens Anders [2], Francisco Bas [1], Steffen Becker [2], Elke De Mulder [3], Nourhan Elhamawy [2], Frank K. Gürkaynak [4], Helena Handschuh [3], Carles Hernandez [5], Mike Hutter [3], Leonidas Kosmidis [1], Ilia Polian [2], Matthias Sauer [6], Stefan Wagner [2], Francesco Regazzoni [7,8]

[1]*Barcelona Supercomputing Center (BSC)*, jaume.abella@bsc.es, sergi.alcaide@bsc.es, leonidas.kosmidis@bsc.es, francisco.basjalon@bsc.es

[2]*University of Stuttgart*, jens.anders@iis.uni-stuttgart.de, steffen.becker@informatik.uni-stuttgart.de, nourhan.elhamawy@iti.uni-stuttgart.de, ilia.polian@informatik.uni-stuttgart.de, stefan.wagner@iste.uni-stuttgart.de

[3]*Rambus*, mhutter@rambus.com, Helena.Handschuh@cryptography.com, Elke.DeMulder@cryptography.com

[4]*ETH Zurich*, kgf@ethz.ch

[5]*Universitat Politecnica de Valencia (UPV)*, carherlu@upv.es

[6]*Advantest Europe*, matthias.sauer@advantest.com

[7]*University of Amsterdam*, f.regazzoni@uva.nl

[8]*Università della Svizzera italiana*, francesco.regazzoni@usi.ch

***Abstract*—RISC-V has emerged as a viable solution on academia and industry. However, to use open source hardware for safety-critical applications, we need a deep understanding of the way in which well established mechanisms for testing and reliability could be integrated and deployed on the RISC-V ecosystem, and we need a clear knowledge on how such an ecosystem can be leveraged to improve security. This paper includes four contributions presenting the potential of RISC-V in security research, the way in which RISC-V can be hardened against power analysis attacks, how to implement, using RISC-V, software and hardware/software solutions for dual core lock step, and how to perform system-level testing in the RISC-V ecosystem.**

***Index Terms*—RISC-V, Security, Side Channel Attacks, Reliability, Testing**

I. Introduction

RISC-V has emerged as a viable solution on academia and industry. Pushed by the success of RISC-V, many other open source hardware projects have started and it is certainly to be expected that even more of them will start in the near future. However, to use open source hardware for safety-critical applications, we need a deep understanding of the way in which well established mechanisms for testing and reliability could be integrated and deployed on the RISC-V ecosystem, and we need a clear knowledge on how such an ecosystem can be leveraged to improve security. To this end, this paper includes four contributions and addresses issues related with the use of RISC-V based architecture in the context of security, reliability and testing.

The first contribution, provided by Frank Gürkaynak from ETH Zurich, summarizes the reasons behind the success of RISC-V and discusses its potential as a platform for researching and developing secure solutions. Among the reasons that makes RISC-V a suitable platform for security research there are: the community that is supporting it and the way in which the governance is organized, the fact it was designed and developed to be easily extended, and the fact that it is accessible.

The second contribution, provided by Elke De Mulder, Helena Handschuh and Mike Hutter from Rambus, after a short introduction on side-channel attacks, presents an overview of the state of the art industrial and academic approaches for making RISC-V robust against these attacks. Reported approaches include gate-level masking in hardware, software only based solutions, instruction set extensions for DPA-resistant coprocessors and protocol level solutions. The contribution concludes discussing testing strategies for assessing the robustness achieved by the protections.

The third contribution, provided by Jaume Abella, Sergi Alcaide, Francisco Bas and Leonidas Kosmidis from Barcelona Supercomputing Center (BSC) and by Carles Hernandez from Universitat Politecnica de Valencia (UPV) focuses on safety-related systems that require high integrity. It presents a RISC-V based implementation of a dual core LockStep system. In these systems, two cores provide redundancy with sufficient independence, but the the number of user-visible cores is halved. This issue is addressed by SW-only and HW/SW solutions to enable diverse redundancy without needing strict dual core LockStep.

The fourth contribution, provided by Jens Anders, Steffen Becker, Nourhan Elhamawy, Ilia Polian, and Stefan Wagner from University of Stuttgart, and by Matthias Sauer from Advantest Europe, describes the specific challenges in making system level testing part of an open ecosystem enabled by the RISC-V instruction set, where some of the information required may not exist. The contribution will present the first results on coverage achieved by structural and system-level test approaches on RISC-V processor and its submodules.

II. Potential of RISC-V in security research

As an open instruction set architecture (ISA), RISC-V is attracting considerable amount of attention both from the industry and academia. This popularity is also mirrored in security research where an increasing number of publications and projects are built around RISC-V based solutions. This despite the fact that RISC-V is by far not the first available open-source ISA, as of 2021 does not yet have a market share to compete with more established commercial ISAs like ARM and Intel, and arguably does not inherently offer added security that can not be replicated by other ISAs. It is therefore important to discuss what makes RISC-V suited for security research.

A. RISC-V has a thriving community

One common misconception around open source is that being *open* is its key ingredient. This unfortunately is not true. There are far too many open source projects, most of which do not enjoy the type of popularity and reach of (as an example) Linux, GNU C compiler, Apache, and RISC-V. What makes an open source project, relevant and attractive is actually its user base.

An open source project gains popularity not only through technical merit, but also through its governance and its main supporters. RISC-V was originally developed as an internal project of the University California Berkeley between 2010-2015 [1]. In 2015, the RISC-V foundation, with 36 founding members (that included large corporations like Google, IBM, Nvidia, Qualcomm and Western Digital), took over the maintenance of the ISA description [2]. The foundation, and its broad support from the industry allowed RISC-V to get a considerable head start and contributed to its popularity.

It is also important to note that, the RISC-V foundation, and later RISC-V International does not manage or make available any open-source RISC-V implementations but only manages the standard specifications. This turned to be a key point in the success RISC-V. The ISA essentially ties the efforts between the software infrastructure (compilers, emulators, operating system ports, development tools) and hardware implementations together, but does not dictate how each can be implemented, significantly reducing disagreements on implementation details. In addition, while the ISA is free, it allows processors implementing the RISC-V ISA to be designed without licensing restrictions thereby allowing both open source and commercial implementations.

The last factor in the popularity of RISC-V is undoubtedly the timing. The continuous development in microelectronics industry has resulted in a phenomenal degree of integration where System-on-Chips (SoC) that feature several processor cores have become commonplace. This relegated the processors from the *main part* of high-end IC design to a *commodity* building block. The RISC-V ISA and its early implementations were available just at the time where individual processor cores were both more in demand, and their individual technical properties started to be less critical. Many projects, products needed *a* processor here and there to do the job, and RISC-V was increasingly up to the task more of the time than less. As a result, in 10 years, under the stewardship of RISC-V International RISC-V has amassed a large community that remains very actively involved.

B. RISC-V has been developed with extensions in mind

From the beginning RISC-V has put a large emphasis on being flexible and extensible. The ISA (like some other commercial ISAs) is designed around a *base* ISA that contains only a handful of instructions. This can be *extended* by additional modules. RISC-V defines actually three families of this *base* ISA for 32, 64 and 128 bit architectures. The additions (usually with a single letter to describe them) can be combined to form more capable architectures, so for example RV32IMCF, defines a processor with the 32 bit **I**nteger base with extensions for **M**ultiplication, **C**ompressed instructions and single-precision **F**loating point support.

All extensions are being governed by working groups and are *ratified* by RISC-V members after a review process. While the process is open and transparent, it takes quite a bit of time until an extension is ratified. As an example, the vector extension for RISC-V was proposed in 2015, and the official v1.0 version is still being refined [3].

But what makes RISC-V more flexible is the fact that portions of the instruction encoding space have been reserved for user extensions. Users are free to develop their own instructions, and as long as they use the reserved space, their implementation will maintain compatibility with standard RISC-V. This allows users to experiment with various extensions while continuing to benefit from the RISC-V ecosystem. As a recent example, the authors in [4] suggest and evaluate the use of an additional `fence.t` instruction to flush all state holding micro-architectural components to mitigate timing-channel attacks in a 64 bit RISC-V core. Such experiments and results are then used in working groups to argue and in some cases ratify new and/or modified extensions.

Since there is no official or canonical RISC-V implementations, RISC-V is not encumbered with implementation decisions that affect the rest of the system. For example, replacement policies used for caches (or even the memory hierarchy), the type of branch prediction or the interconnect subsystem are not part of the RISC-V ISA description. This has allowed many completely different RISC-V implementations that cover a much larger micro-architectural space than any other commercial ISA. In other words, RISC-V has to put effort to remain flexible enough to support the diversity of its implementations, which make it much more amenable to be adapted and extended in new ways.

C. RISC-V is accessible

As described above, RISC-V has a large user base, a stable governing body and support from major industrial and academic users. In a very short time frame RISC-V has risen from a tool for teaching computer architecture classes to a well-liked and used processor system that has established firmly itself.

As an open source ISA, there are many available RISC-V implementations that are being made available as open source hardware (as opposed to many commercial RISC-V offerings), some with significant industry backing such as those from Western Digital [5], OpenHW Group [6], LowRISC [7], and some popular implementations by UC Berkeley [8], IIT Madras [9], ETH Zurich [10], Charles Pappon [11] and Claire Wolf [12], covering a wide range of implementation languages, styles and targets. This diversity ensures that no matter what the background and application field, it is possible to find a suitable, mature open source hardware implementation of RISC-V to use.

In fact, the *openness* of RISC-V implementations has proven to be one if its main assets. Even commercial companies that develop their own proprietary processors realize that concepts and ideas can easily be demonstrated on an open RISC-V platform. This allows companies to co-operate with others without divulging any company secrets. Development and evaluation is performed on an open RISC-V platform that is unencumbered by any restrictions. Once the viability of the idea has been demonstrated, the company can then continue to adapt the techniques internally on their own architectures.

Open hardware platforms have also provided researchers, especially in the security field, unprecedented access to the internals of the processor. Implementation details of processors that remained hidden, either intentionally or due to lack of documentation became accessible to all that are reasonably proficient with the languages that the open hardware has been described. This has facilitated systematic analyses of the systems, rather than ad-hoc methods that had to *guess* how certain functions were implemented.

Of course, there is the most obvious asset that open source RISV-V implementations have provided security researchers: they have the ability to make any and every change to a working processor and directly evaluate the efficacy of their proposed changes and compare the implementation cost directly. Instruction set extensions, additional accelerators, adding checksums, tags for control flow integrity, every possible countermeasure, hardening feature can directly be implemented on the actual running system and evaluated.

In summary, The major attraction of RISC-V as vehicle for research in computer architecture related fields is actually the combination of these three effects. It has a large user base, it is designed to be extendable, is not tied to a specific implementation, and a diverse set of mature open source implementations are available for researchers to use.

III. Thwarting Differential Power Analysis Attacks on RISC-V processors

Side-channel attacks (SCA) on cryptographic implementations are a class of attacks which make use of physical information collected from a device executing said implementation. We will only focus on passive side-channels. The most common are power, timing and electromagnetic radiation. Those easily measurable quantities depend on what is happening inside the device under test (DUT). This entails that in the case where a cryptographic implementation is running on the DUT, the side-channels contain information on the internal state of the algorithm, therefore breaking the original cryptanalytic assumption that the only information an attacker has at hand are inputs and/or outputs to an algorithm. Indeed, the side channels contain information about the intermediate states of the cryptographic block throughout the entire computation and those intermediate states typically only depend on a partial key instead of the full key like the output would. A typical side-channel attack combines side-channel information, intermediate state values, cryptanalytic and statistical techniques to verify guesses of a partial key in timing, simple power analysis (SPA) or differential power analysis attacks (DPA) [13].

Software (SW) as well as hardware (HW) implementations are vulnerable to Side-channel Analysis (SCA) attacks, but protecting software implementations has the additional difficulty in that they run on processors of which the implementation is typically unknown to the developer. This can lead to a whole array of, at first sight, inexplicable leakage. It is a problem which is being studied in academia [14]–[21], but a generic, provable solution for the problem has not been found yet. While some of the leakage can be explained with knowledge of the ISA of the processor, other leakage can only be explained by knowing the exact micro-architectural implementation of the processor and the exact sequence of low level code which is being executed. The leakage can occur inside or be caused by all processor elements that handle secret data including, but not limited to the Arithmetic-Logic-Unit (ALU) in the data path, the register bank, internal and external memory, the control path, the branching unit, JTAG etc. All components need to be analyzed and protected carefully. In the following section we will exclude leakage caused by speculative execution, the so called Meltdown and Spectre-like attacks [22], [23] and focus on the traditional DPA style attacks. Similarly, we will exclude fault attacks, for more information on those see o. a. [24].

A. Problem description

There are several classes of leakage, for the sake of simplicity we distinguish three different types: direct-value, value-combination (overwrite leaks) and circuit-level leakage. By using masking, a well-known countermeasure against passive SCA, where, rather than doing calculations on real values, the values are split under some operand into uniformly distributed shares and the calculations are performed on those shares in such a way that the result can be reconstructed afterwards, direct value leaks are avoided. A common masking scheme is d^{th}-order Boolean masking, where d shares combined with the Boolean XOR operation create the real value, x being processed, $x_1 \oplus x_2 \oplus \ldots \oplus x_d = x$.

Value combination leakage occurs when all shares of a secret get combined in the processor. A simple example is the overwrite of the two shares of a 2^{nd} order Boolean masking scheme in the register bank, but this process can occur at multiple other locations, like data transfers on a

bus. A typical countermeasure is to randomly precharge or clear registers as in [25]. Since the actual flow of a share throughout the processor is not known, even a carefully written software implementation cannot always avoid this effect and as a consequence code cannot be ported easily between CPUs.

A third class of leakage is circuit-level leakage, at the gate level. In CMOS, a Boolean gate usually does not only make a bit transition in every clock cycle but the gate output can toggle several times depending on the value and signal arrival-times of the inputs. These *glitches* are a substantial contributor to the dynamic power consumption and they can create strong leakage. Masking techniques are necessary that split up the computation into independent processing *streams* or *domains*. Examples are techniques that are based on secure Multi-Party Computation (MPC) or Threshold Implementations (Consolidated Masking Schemes, Domain-oriented Masking, etc.). Any cross-domain link (or interconnection) between hardware components that stores or processes all shares of a secret will cause leakage.

A pure software solution can become quite expensive due to the need of a high order masking scheme or due to a significantly larger design time when the implementation needs to be small and fast. To demonstrate the challenge of creating a software only solution, we will describe some of the leakage which can occur by a simple Boolean masked AND operation with 2 shares as described in Alg. 1.

Algorithm 1 Boolean Masked AND

Require: Boolean masked value X as (X_0, X_1) such that $X_0 \oplus X_1 = X$ and value Y as (Y_0, Y_1) such that $Y_0 \oplus Y_1 = Y$
Ensure: (Z_0, Z_1) such that $Z_0 \oplus Z_1 = Z$ with $Z = X \wedge Y$
1: Generate a random value M
2: $Z_0 = Z_1 = M$
3: $T = X_0 \wedge Y_0$
4: $Z_1 = Z_1 \oplus T$
5: $T = X_1 \wedge Y_0$
6: $Z_1 = Z_1 \oplus T$
7: $T = X_0 \wedge Y_1$
8: $Z_1 = Z_1 \oplus T$
9: $T = X_1 \wedge Y_1$
10: $Z_1 = Z_1 \oplus T$

Every intermediate is independent of the secret values X,Y or Z, we therefore do not expect direct-value leaks. But taking a look at which values are consecutively computed upon by the AND logic part of the CPU, one can see that X_0 in Step 3 and X_1 in Step 5 are consecutively used as the left side of an AND operation which means that we have to see whether the Hamming distance of these values are not related to any secret. Unfortunately, the Hamming distance equals X which is a secret, hence the algorithm will have to be rewritten to take this into account. A little less straightforward is that the values of T in Steps 3 and 5 are not entirely decorrelated from the value X because of the use of the AND operation which means that updating the register associated with consecutive T values will cause side-channel leaks. In this case, the software developer will have to keep this in mind and find a suitable solution. The end result is either an algorithm which will require significantly more instructions and/or the usage of more registers or memory, thereby wasting important resources. On top of this, the development cycle will be longer because every change in software needs to be tested against side-channel leakage again.

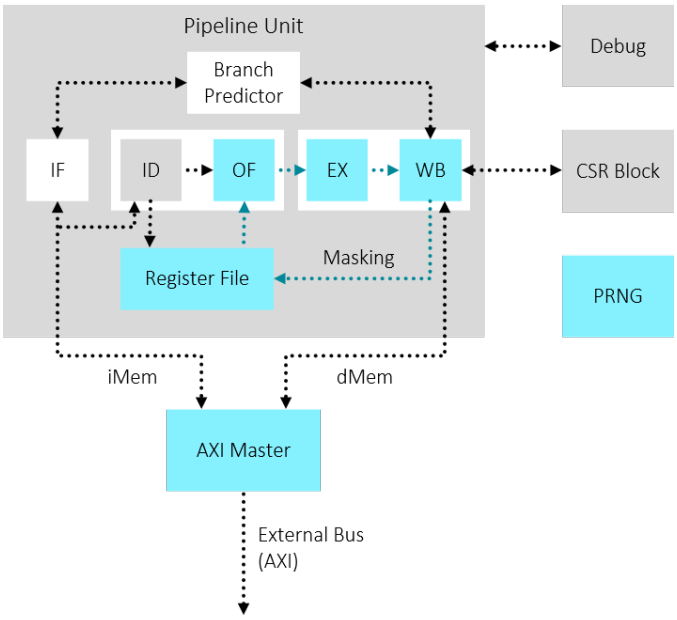

Fig. 1. SCA-hardened processor from [28].

B. Solutions in literature

In [26] the authors describe the design of a SCA-hardened and flexible processor design based on the open-source V-scale RISC-V processor, a single-issue in-order 3-stage pipeline design. The processor can be synthesized to protect against SCA attacks of arbitrary attack order. The solution describes how the ALU operations are protected by using domain-oriented masking (DOM). DOM is introduced in [27]. The 3-stage pipeline is transformed into a 4-stage pipeline by creating a separate decode and execute stage. Care has been taken that there is no accidental leakage through the addition of merging buffers. A PRNG is added for the necessary randomness. The memory interface is not protected. For efficiency reasons, the division/multiplication unit, address calculation, and data comparison for conditional jumps is not protected.

In [28], an instance of an SCA-hardened processor is introduced based on a single-issue, in-order, 5-stage pipeline architecture. A graphical representation of the processor is given in Fig. 1. The blue elements are the parts which have been altered to ensure DPA resistance. In contrast to [26] the memory interface is also protected. This solution is designed to be $1st$-order DPA resistant and doubles the register file. The ALU is protected similarly to [26] but uses a $d+1$ Threshold Implementation [29] instead of DOM.

The authors of [30] follow a more heuristic approach. They devised a flow which characterises and pinpoints the most leaking modules in a processor, they demonstrate the principle

on RISC-V. After identification, they mitigate the leakage by introducing a set of countermeasures without relying on traditional masking techniques. Instead their countermeasures vary from simple modifications of the HDL code ensuring secure translation by the EDA tools, to obfuscating data and address lines. Their goal is to reduce the leakage discovered during the initial detection phase, therefore, while their design approach can be applied to a wide variety of processors, the reduction of the leakage is only partial due to the limitations of the initial profiling phase.

A DOM ISA is proposed in [31] where a protected and unprotected datapath coexist in the same processor and depending on the sensitivity of the operations one or the other can be chosen by the developer. To eliminate leakage the two datapaths are kept completely separate. Earlier work [31] provided custom instructions that support masking as well as bitslicing and fault detection, all of which could be layered. However, to use it, the software developer was burdened with bitslicing the code and had to pay special attention in allocating registers for their variables.

C. Testing

Testing a DPA-secure processor is not a trivial task. The more complex a processor, the more modules and code sequences can create leakage. As with any testing, covering all (corner) cases is a difficult task. Most of the papers listed here use a form of Test Vector Leakage Assessment [32] (TVLA), namely fixed-versus-random input testing. Since testing for these processors is not yet standardized, it still requires quite some knowledge and expertise from the developer, but the TVLA framework provides a first step in easing the job and is already being used by many organizations including certification labs and development teams to assess the most well-known cryptographic algorithm implementations against side-channel leakage.

IV. SW-only and HW/SW support for diverse redundancy for high-integrity applications

The increasing popularity of RISC-V brings abundant opportunities to leverage hardware IP and software tools from the RISC-V ecosystem for their use in many domains. Due to this, industry for high-integrity and/or edge systems has started developing their own hardware/software (HW/SW) platforms on RISC-V [33]–[35]. However, whenever those systems inherit some safety requirements, specific support to implement safety measures efficiently is needed. For the highest criticality functionalities, those safety measures include some form of diverse redundancy so that a single fault, despite affecting all redundant elements, cannot lead to exactly the same error, which might escape detection. However, RISC-V platforms available do not offer support for diverse redundancy yet. In this section, we present some ongoing research activities aiming at closing this gap with so-called light lockstep solutions. First, we provide some background on the problem of diverse redundancy, then we introduce SW-only solutions, and finally we present HW/SW approaches aiming to deliver such diverse redundancy.

A. Background on Diverse Redundancy

Common Cause Failures (CCFs), in automotive terminology [36], are those failures caused by a single fault that makes safety measures, such as redundancy, ineffective. For instance, two identical cores executing the same task redundantly fully synchronized have the same state, and upon a common fault (e.g. a voltage droop) could experience the same error. To avoid CCFs, safety-related systems implement redundancy with some form of diversity so that the risk of experiencing identical errors in redundant elements is residual. In the case of storage, this is usually achieved using Error Detection Codes, in the case of communications using Cyclic Redundancy Check codes, and in the case of computation using some form of lockstepped execution where two or more identical cores execute identical software but with some staggering (i.e. time shift) so that cores' state is sufficiently diverse.

To allow RISC-V platforms being adopted in safety-related systems, some form of lockstepping support is needed. This can be achieved with tight lockstepping, where only one redundant core is visible at software level and the others can only work in lockstep mode, as done, for instance, by the Infineon AURIX microcontrollers for the automotive domain. Tight lockstepping at core level can be implemented with different flavors. For instance, one could compare the outcome of each instruction or even each pipeline stage every cycle. However, the most effective solution has been shown to compare only off-core activity (e.g. requests visible in the interconnect) to reduce the overheads while avoiding any visible impact due to errors. This solution has been further extended with a periodic dump of the register file contents to limit the time since an error occurs until it is detected to ease the implementation of simpler recovery means [37].

While this solution is highly effective to attain diverse redundancy, it is inflexible since it does not allow using the cores independently to run different tasks. An alternative is using light lockstepping, where redundancy is created and managed at software level, and independent cores are used enforcing staggering either with SW-only means or with a combination of HW/SW support. This section presents some ongoing research efforts to deliver support for diverse redundancy with light lockstepping, either with SW-only solutions or HW/SW solutions.

B. SW-only Diverse Redundancy

SW-only diverse redundancy builds upon the creation of redundant processes at software level, so that both of them receive the same – redundant – input data, and return their results for comparison in a *safe* CPU. Such a solution has been investigated in [38] in the context of a platform combining a moderate performance CPU with native (tight) lockstep support, and a high-performance multicore based on Arm cores without any HW lockstep support. Without loss of generality, the rest of the section focuses on dual-core lockstep solutions,

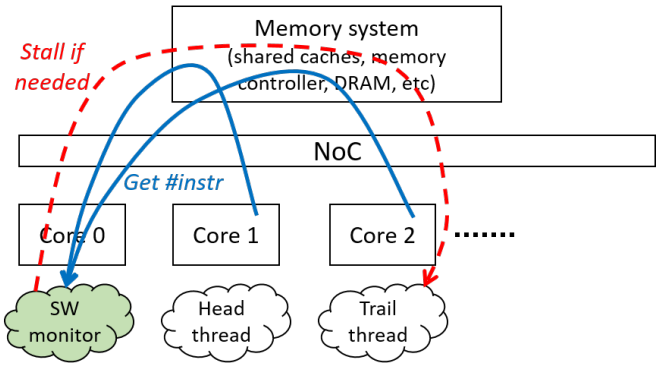

Fig. 2. Schematic of the SW-only lightweight lockstep.

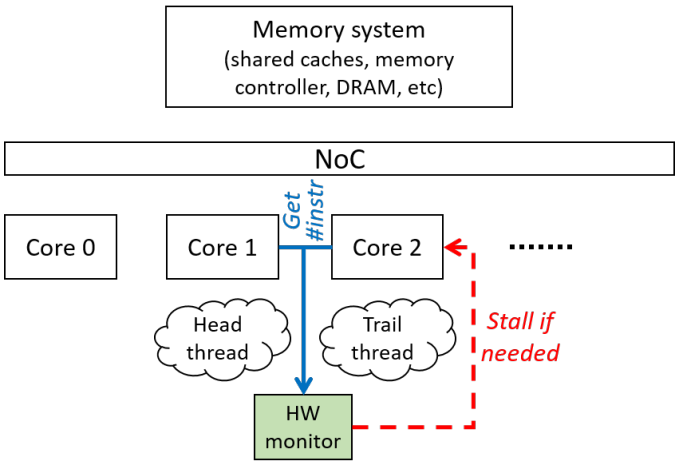

Fig. 3. Schematic of the HW/SW lightweight lockstep.

although analogous reasoning applies for any other number of cores. The solution builds on creating three threads (see Figure 2):

- **Monitor**. The monitor is the one spawning the redundant computation threads (head and trail), and monitoring and enforcing staggering between them.
- **Head thread**. The head thread executes the functionality without any specific control for the sake of achieving diverse redundancy.
- **Trail thread**. The trail thread executes the functionality redundantly with some staggering (delay) w.r.t. the head thread. Therefore, if the staggering at any point is too short, it is stalled for a while.

Since the monitor lacks any form of redundancy, it needs to run in a native lockstepped core, which may be in a separate microcontroller, as in [38], or may also be in the same microcontroller. The monitor spawns the head and trail threads into two other cores, which do not implement tight lockstepping support. Then, the monitor performs the following steps periodically every T_{check} cycles:

1) Collects the instructions executed count (IC) from both threads, so IC_{head} and IC_{trail}.
2) Computes the difference between both counters and compares it against a threshold $I_{stagger}$.
   a) If the head thread is sufficiently ahead from the trail one, then both threads continue the execution. Formally, execution continues normally if $(IC_{head} - IC_{trail}) \geq I_{stagger}$.
   b) Else, if the trail thread is too close to the head one, then the monitor stalls the trail thread during the next monitoring period (so T_{check} cycles). Formally, the trail thread is stalled if $(IC_{head} - IC_{trail}) < I_{stagger}$.
3) Finally, the monitor sleeps until the remaining time until elapsing T_{check} cycles.

This mechanism guarantees that the trail thread cannot catch up with the head thread as long as the instruction threshold $I_{stagger}$ is large enough so that, even if the head thread stalls completely and the trail thread executes at the maximum possible speed, the trail thread cannot catch up with the head thread since one monitoring period until the next one. This implies that executing $I_{stagger}$ instructions must require at least T_{check} plus the time to detect that the current staggering is too low and stalling the trail thread. Therefore, there is a direct relationship between both values, which must be traded off. In particular, the higher T_{check}, the lower the monitoring overhead since the monitor executes less often. However, this makes $I_{stagger}$ grow, so that the overall execution time will increase since, once the head thread finishes its execution, the trail thread still has to execute at least $I_{stagger}$ instructions. Due to the time required to collect the instruction counts from the cores where the head and trail threads execute, and to stall the trail thread if needed, it has been shown that T_{check} should typically correspond to between 100μs and 1ms.

This solution, which so far has been proven doable on Arm-based architectures [38] is currently being ported onto RISC-V platforms in the context of the ECSEL FRACTAL project [35][1], targeting edge systems with some form of safety requirement.

C. HW/SW Diverse Redundancy

While the SW-only solution can be used on RISC-V multicores without any specific hardware support, it imposes some constraints in terms of staggering that make it poorly efficient for short tasks. For instance, if a task requires less than T_{check} cycles to execute, the SW-only solution imposes full serialization of the head and trail threads. Hence, it T_{check} matches 1ms, tasks taking less than 1ms are fully serialized and their overall execution time doubles.

To tackle this limitation, HW/SW solutions can be developed where, instead of having a software monitor polling the other cores via software to collect their instruction counts, and sending a stop signal to the trail thread via software, this process is implemented by hardware means in a hardware

[1]This work has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 877056. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Spain, Italy, Austria, Germany, France, Finland, Switzerland.

monitor module. The HW monitor has direct access to the instruction counts from the two cores being monitored, and reads those values without interfering with the execution of the cores. If the staggering is too low, then it can simply stall the pipeline of the trail thread with the appropriate hardware signal. Since this process can be performed in a very short time and without interfering the execution of the head and trail threads, T_{check} can be set typically to tens or hundreds of cycles, thus typically imposing a staggering of less than 1μs.

Moreover, by performing the monitoring task in a specific module, monitoring does not need an extra core. On the other hand, such module requires internal diverse redundancy to avoid CCFs in the monitor, so it must be implemented following the same principles as tight lockstepped cores. However, such module is expected to be tiny, thus leading to negligible relative cost.

Finally, if the HW monitor is extended with capabilities to compare the outputs from the redundant execution, then the need for a native tight lockstepped core is fully removed.

This solution is currently being developed in the context of a RISC-V platform for safety-critical systems in the context of the H2020 SELENE project [34][2], targeting safety-related space, railway and automotive systems.

V. Towards Bridging the Gap between System-level and Structural Test for the RISC-V Platform

Microprocessors implementing the open RISC-V instruction set and systems-on-chip on their basis are increasingly being considered for safety-critical applications. A prerequisite for their acceptance in domains such as automotive is the feasibility of test procedures established in such domains. System-level test (SLT) is widely employed by both semiconductor manufacturers and system integrators to guarantee the required quality levels, based on procedures, data formats and protocols that have grown over time and are supported by multiple parties. A transition from established platforms to an open RISC-V ecosystem comes with several challenges. In this section, we will discuss these challenges, present first ideas for solutions, and outline further opportunities for system-level test in the RISC-V context.

A. Manufacturing Test: From Structural to SLT

Following the unprecedented growth of sizes and complexities of integrated circuits (IC) over the last decades, the *structural test* approach has become predominant: A list of faults according to a fault model is generated, and for each of these faults, its non-presence in the circuit is verified by applying a *test pattern* that detects it. For sufficiently simple fault models such as stuck-at faults, this approach keeps the complexity of test generation and test application in check, as the number of faults is linear in the size of the circuit. Structural test normally makes use of design-for-test (DFT) circuitry such as scan chains or test points, further reducing its complexity.

System-Level Test (SLT) [40] follows a different principle. The circuit under test is mounted on a board that mimics the intended usage of the circuit in its actual application, such as a smartphone or an electronic control unit of a vehicle. The board includes memories and peripherals that are usually not available during structural test using automated test equipment (ATE). This allows the test engineer to use functional workloads, such as operating system boot or apps that would normally run on the target system, as test cases. SLT has been reported to detect failing ICs that slip through conventional testing [39]. We currently do not know with certainty what leads to such "SLT-unique fails", but possible causes are unmodeled defects (e.g., "soft" timing errors), systematic ATPG coverage holes and system-level effects such as software-controlled clock- and power-domain interactions [39].

Fig. 4 visualizes the role of SLT in a typical test flow. Most companies use at least two test insertions before SLT: wafer sort, where tests are applied to circuits on an undiced wafer, and final test, applied to packaged chips. It is desirable to identify a defective circuit during an as early test insertion as possible to avoid the cost of, e.g., packaging a die that is non-functional anyway. Test engineers speak of the "rule of ten": if a defective circuit escapes detection during a given test insertion, the unnecessary costs for its further processing and testing increases by roughly one order of magnitude (this "rule of ten" extends into test escapes being delivered to the customer).

SLT comes with significant costs. It has very long application times that can go into minutes [41], [42]. As a consequence, it is difficult to incorporate SLT into a structural testing framework, as determining by simulation which faults have been detected during SLT would lead to a very long simulation time. Moreover, scan chains and other DFT are not used during SLT, and faults manifest themselves by unexpected behavior patterns such as crashes or wrong outcomes of a computation, complicating even the decision whether a fault has been detected or not. For the same reason, root-cause diagnosis becomes harder, necessitating machine-

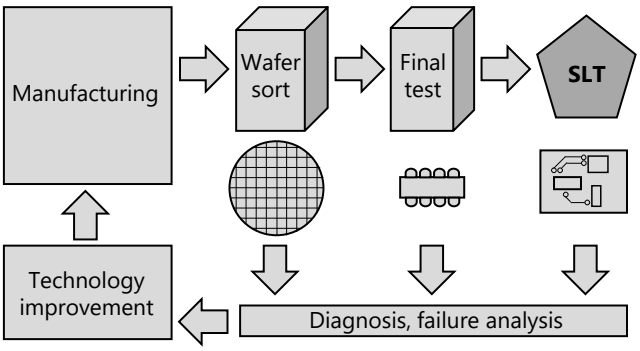

Fig. 4. Structural test and SLT within the quality-assurance flow [39]

[2]This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no. 871467.

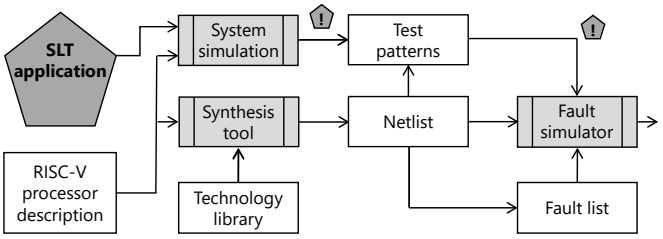

Fig. 5. SLT fault coverage evaluation flow; pentagons mark SLT-specific steps

learning approaches [43]. These costs are offset by its being so close to actual system operation and its ability to identify "test escapes" that would otherwise have been delivered to the customer. In fact, SLT often serves as a "quality gate-keeper" to estimate how well previously applied structural tests work.

B. Towards SLT for RISC-V Processors

The conventional structural test is well supported by electronic design automation (EDA) tools. State-of-the-art RISC-V implementations are compatible with standard design flows, which enables a straightforward use of conventional structural test procedures and design-for-testability approaches. In contrast, SLT is more tightly bound to the specific processor platform and typically makes use of features not readily available in standard EDA tools. One SLT-related problem is to estimate the fault coverage obtained when running software, managed by an operating system, on the processor.

Figure 5 shows our flow for fault coverage analysis of arbitrary SLT programs in a RISC-V processor. It goes through the usual synthesis steps, yet the test patterns for each block within the circuit are not obtained from an automatic test pattern generation (ATPG) tool, but rather from the outcomes of system-level simulation of the whole processor model with an OS and apps running on it. Such simulation generates waveforms for internal signals within the processor. To calculate the fault coverage in a specific module, we match the inputs of that module within the processor's netlist with system-wide signals and record the waveforms on these signals in an extended Value Change Dump (eVCD) file. The values from such eVCD files are then translated into the format which the ATPG tool can read, and the ATPG tool is invoked in fault-simulation mode, taking external test patterns rather than generating them.

We applied the flow of Figure 5 to the CVA6 RISC-V CPU [6] and several simple software applications. To perform system-level simulation of an application, we compiled it for the CVA6 CPU, resulting in an ".elf" file, and loaded this file into the main memory of the CVA6 CPU in the RTL simulation model. Running the simulation generates eVCD files that are, after a conversion, handed over to one of the standard commercial ATPG tools.

Table I shows first results obtained for various submodules of the CVA6 core under four different programs: the Blinky demonstration of the FreeRTOS, the "Hello,World" program, the Minimal FreeRTOS boot and the Dhrystone benchmark. The table shows, for each submodule, the total number of stuck-at faults, the number of undetectable faults (as reported by the ATPG tool), and the fault and fault efficiency (percentage of detected faults over all detectable faults). The fault efficiency for patterns generated by the structural stuck-at ATPG tool are shown in the last column of the table for comparison. One sees that the coverage achieved by simple test programs is actually not very high for many of the submodules. This is not unexpected, because these programs are not optimized towards using all functions of the CPU; for instance, they do not invoke all RISC-V instructions. Interestingly, even the ATPG patterns do not reach coverages close to 100% for many submodules. While the results illustrates the applicability of our flow for a detailed evaluation of given programs, it aslo highlights the need for manually crafted or automatically generated SLT software.

C. Potentials of an Open RISC-V Ecosystem for SLT

For an organization with an existing experience in SLT based on a given (closed-source) platform, moving to a new ecosystem will be associated with efforts and costs, simply because solutions that have been created over decades may not have an equivalent in the RISC-V universe. One can expect, however, that such solutions will emerge over time, and will be offered by commercial vendors or even put into the public domain. The open nature of RISC-V provides an opportunity to explore SLT-related problems in depth by many individual players. The availability of widely or universally agreed-upon standards can also simplify the communication between the semiconductor makers, who are using SLT as part of their outgoing quality control, and their customers, i.e., the system integrators, for whom SLT is the cornerstone of incoming quality assurance.

An interesting potential for SLT is given by the extensibility of the RISC-V instruction set. Producers of systems that need diagnostic observability or buit-in self-test (BIST) features (such features can be demanded by functional safety standards) could introduce new instructions dedicated to SLT. For instance, each hardware module within the CPU could be equipped with BIST circuitry that has to be used periodically. With dedicated BIST instructions, the software could schedule such tests itself, leveraging the knowledge which modules will not be needed in the next future.

VI. Conclusions

To enable the use of RISC-V in safety-critical applications, it is necessary to have a complete understanding of the way in which well established mechanisms for testing and reliability could be integrated and deployed on its ecosystem. It is also fundamental to have a clear knowledge on how such an ecosystem can be leveraged to improve security. This paper summarized four contributions addressing issues related with the use of RISC-V based architecture in the context of security, reliability and testing

Acknowledgment

This work is partially supported by Advantest as part of the Graduate School "Intelligent Methods for Test and Reliability"

TABLE I
FAULT EFFICIENCY FOR DIFFERENT SUBMODULES OF CVA6 RISC-V CPU AND DIFFERENT SLT APPLICATIONS

Module	# Faults		Fault efficiency				
	Total	Undetectable	Blinky	Hello	Minimal	Dhrystone	Structural ATPG
commit_stage	6,770	974	23.07	25.08	23.07	23.07	61.62
├─ controller	412	266	19.90	16.99	29.61	20.63	35.44
├─ csr_regfile	60,484	3,593	14.79	15.49	14.79	14.78	91.13
├─ frontend	102,140	510	06.25	06.09	06.25	06.34	50.35
├─ id_stage	13,772	374	19.34	20.72	19.34	19.42	95.80
│ ├─ c.decoder	1,654	0	46.19	86.88	69.95	70.19	100
│ └─ decoder	8,952	372	40.24	44.96	40.26	39.99	78.89
└─ perf_counters	30,936	1,932	05.27	00.00	05.27	05.27	93.75

(GS-IMTR) at the University of Stuttgart and by the European Union Horizon 2020 research and innovation program under CPSoSAware project (grant No. 871738)

REFERENCES

[1] K. Asanović and D. A. Patterson, "Instruction sets should be free: The case for risc-v," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2014-146, Aug 2014. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-146.html

[2] "Founding members - risc-v international," https://riscv.org/membership/founding-members/, (Accessed on 03/22/2021).

[3] R.-V. V. extension working group, "Working draft of the proposed risc-v v vector extension," https://github.com/riscv/riscv-v-spec, 2021.

[4] N. Wistoff, M. Schneider, F. K. Gürkaynak, L. Benini, and G. Heiser, "Prevention of microarchitectural covert channels on an open-source 64-bit risc-v core," 2020.

[5] W. Digital, "Eh2 swerv risc-v core," https://github.com/chipsalliance/Cores-SweRV-EH2, 2020.

[6] F. Zaruba and L. Benini, "The cost of application-class processing: Energy and performance analysis of a linux-ready 1.7-ghz 64-bit risc-v core in 22-nm fdsoi technology," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 11, pp. 2629–2640, Nov 2019.

[7] LowRISC, "Ibex risc-v core," https://github.com/lowRISC/ibex, 2020.

[8] K. Asanović, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz, S. Karandikar, B. Keller, D. Kim, J. Koenig, Y. Lee, E. Love, M. Maas, A. Magyar, H. Mao, M. Moreto, A. Ou, D. A. Patterson, B. Richards, C. Schmidt, S. Twigg, H. Vo, and A. Waterman, "The rocket chip generator," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17, Apr 2016. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.html

[9] S. Das, R. H. Unnithan, A. Menon, C. Rebeiro, and K. Veezhinathan, "Shakti-ms: A risc-v processor for memory safety in c," in *Proceedings of the 20th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems*, ser. LCTES 2019. New York, NY, USA: Association for Computing Machinery, 2019, p. 19–32. [Online]. Available: https://doi.org/10.1145/3316482.3326356

[10] M. Gautschi, P. D. Schiavone, A. Traber, I. Loi, A. Pullini, D. Rossi, E. Flamand, F. K. Gürkaynak, and L. Benini, "Near-threshold risc-v core with dsp extensions for scalable iot endpoint devices," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 10, pp. 2700–2713, 2017.

[11] C. Pappon, "A fpga friendly 32 bit risc-v cpu implementation," https://github.com/SpinalHDL/VexRiscv, 2020.

[12] C. Wolf, "Picorv32 - a size-optimized risc-v cpu," https://github.com/cliffordwolf/picorv32, 2020.

[13] P. C. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, ser. Lecture Notes in Computer Science, M. J. Wiener, Ed., vol. 1666. Springer, 1999, pp. 388–397.

[14] J. Balasch, B. Gierlichs, V. Grosso, O. Reparaz, and F. Standaert, "On the cost of lazy engineering for masked software implementations," in *Smart Card Research and Advanced Applications - 13th International Conference, CARDIS 2014, Paris, France, November 5-7, 2014. Revised Selected Papers*, ser. Lecture Notes in Computer Science, M. Joye and A. Moradi, Eds., vol. 8968. Springer, 2014, pp. 64–81.

[15] Y. L. Corre, J. Großschädl, and D. Dinu, "Micro-architectural power simulator for leakage assessment of cryptographic software on ARM cortex-m3 processors," in *Constructive Side-Channel Analysis and Secure Design - 9th International Workshop, COSADE 2018, Singapore, April 23-24, 2018, Proceedings*, ser. Lecture Notes in Computer Science, J. Fan and B. Gierlichs, Eds., vol. 10815. Springer, 2018, pp. 82–98.

[16] K. Papagiannopoulos and N. Veshchikov, "Mind the gap: Towards secure 1st-order masking in software," in *Constructive Side-Channel Analysis and Secure Design - 8th International Workshop, COSADE 2017, Paris, France, April 13-14, 2017, Revised Selected Papers*, ser. Lecture Notes in Computer Science, S. Guilley, Ed., vol. 10348. Springer, 2017, pp. 282–297.

[17] A. Barenghi and G. Pelosi, "Side-channel security of superscalar cpus: evaluating the impact of micro-architectural features," in *Proceedings of the 55th Annual Design Automation Conference, DAC 2018, San Francisco, CA, USA, June 24-29, 2018*. ACM, 2018, pp. 120:1–120:6.

[18] C. Cernazanu-Glavan, M. Marcu, A. Amaricai, S. Fedeac, M. Ghenea, Z. Wang, A. Chattopadhyay, J. Weinstock, and R. Leupers, "Direct fpga-based power profiling for a RISC processor," in *2015 IEEE International Instrumentation and Measurement Technology Conference (I2MTC) Proceedings, Pisa, Italy, May 11-14, 2015*. IEEE, 2015, pp. 1578–1583.

[19] M. A. Shelton, N. Samwel, L. Batina, F. Regazzoni, M. Wagner, and Y. Yarom, "Rosita: Towards automatic elimination of power-analysis leakage in ciphers," *CoRR*, vol. abs/1912.05183, 2019.

[20] W. Diehl, A. Abdulgadir, and J. Kaps, "Vulnerability analysis of a soft core processor through fine-grain power profiling," *IACR Cryptol. ePrint Arch.*, vol. 2019, p. 742, 2019.

[21] L. De Meyer, E. De Mulder, and M. Tunstall, "On the effect of the (micro)architecture on the development of side-channel resistant software," *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 1297, 2020.

[22] P. Kocher, J. Horn, A. Fogh, , D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," in *40th IEEE Symposium on Security and Privacy (S&P'19)*, 2019.

[23] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown: Reading kernel memory from user space," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018.

[24] M. Werner, R. Schilling, T. Unterluggauer, and S. Mangard, "Protecting RISC-V processors against physical attacks," in *Design, Automation & Test in Europe Conference & Exhibition, DATE 2019, Florence, Italy,*

March 25-29, 2019, J. Teich and F. Fummi, Eds. IEEE, 2019, pp. 1136–1141.

[25] S. Tillich, C. Herbst, and S. Mangard, "Protecting aes software implementations on 32-bit processors against power analysis," in *ACNS*. Springer-Verlag, 2007, pp. 141–157.

[26] H. Groß, M. Jelinek, S. Mangard, T. Unterluggauer, and M. Werner, "Concealing secrets in embedded processors designs," in *Smart Card Research and Advanced Applications - 15th International Conference, CARDIS 2016, Cannes, France, November 7-9, 2016, Revised Selected Papers*, ser. Lecture Notes in Computer Science, K. Lemke-Rust and M. Tunstall, Eds., vol. 10146. Springer, 2016, pp. 89–104.

[27] H. Groß, S. Mangard, and T. Korak, "Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order," in *Proceedings of the ACM Workshop on Theory of Implementation Security, TIS@CCS 2016 Vienna, Austria, October, 2016*, B. Bilgin, S. Nikova, and V. Rijmen, Eds. ACM, 2016, p. 3.

[28] E. De Mulder, S. Gummalla, and M. Hutter, "Protecting RISC-V against side-channel attacks," in *Proceedings of the 56th Annual Design Automation Conference 2019, DAC 2019, Las Vegas, NV, USA, June 02-06, 2019*. ACM, 2019, p. 45.

[29] S. Nikova, V. Rijmen, and M. Schläffer, "Secure hardware implementation of nonlinear functions in the presence of glitches," *J. Cryptol.*, vol. 24, no. 2, pp. 292–321, 2011.

[30] M. A. K. F, V. Ganesan, R. Bodduna, and C. Rebeiro, "PARAM: A microprocessor hardened for power side-channel attack resistance," in *2020 IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2020, San Jose, CA, USA, December 7-11, 2020*. IEEE, 2020, pp. 23–34.

[31] P. Kiaei, D. Mercadier, P. Dagand, K. Heydemann, and P. Schaumont, "Custom instruction support for modular defense against side-channel and fault attacks," in *Constructive Side-Channel Analysis and Secure Design - 11th International Workshop, COSADE 2020, Lugano, Switzerland, April 1-3, 2020, Revised Selected Papers*, ser. Lecture Notes in Computer Science, G. M. Bertoni and F. Regazzoni, Eds., vol. 12244. Springer, 2020, pp. 221–253.

[32] J. Jaffe, G. Goodwill, B. Jun, and P. Rohatgi, "A testing methodology for side-channel resistance validation," 2011. [Online]. Available: http://csrc.nist.gov/news_events/non-invasive-attack-testing-workshop/papers/08_Goodwill.pdf

[33] De-RISC Consortium, "De-RISC: Dependable Real-Time Infrastructure for Safety-Critical Computer," 2021, https://www.derisc-project.eu/ (accessed Mar-2021).

[34] SELENE Consortium, "SELENE: Self-Monitored Dependable Platform for High-Performance Safety-Critical Systems," 2021, https://www.selene-project.eu/ (accessed Mar-2021).

[35] FRACTAL Consortium, "FRACTAL: Cognitive Fractal and Secure EDGE Based On Unique Open-Safe-Reliable-Low Power Hardware Platform Node," 2021, https://fractal-project.eu/ (accessed Mar-2021).

[36] International Standards Organization, *ISO/DIS 26262. Road Vehicles – Functional Safety*, 2009.

[37] C. Hernandez and J. Abella, "Timely error detection for effective recovery in light-lockstep automotive systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 11, pp. 1718–1729, 2015.

[38] S. Alcaide, L. Kosmidis, C. Hernandez, and J. Abella, "Software-only based diverse redundancy for asil-d automotive applications on embedded hpc platforms," in *2020 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2020, pp. 1–4.

[39] I. Polian *et al.*, "Exploring the mysteries of system-level test," in *IEEE Asian Test Symp.*, 2020.

[40] H. H. Chen, "Beyond structural test, the rising need for system-level test," in *VLSI-DAT*. IEEE, 2018, pp. 1–4.

[41] A. D. Singh, "An adaptive approach to minimize system level tests targeting low voltage DVFS failures," in *ITC*. IEEE, 2019, pp. 1–10.

[42] P. Bernardi, M. Restifo, M. S. Reorda, D. Appello, C. Bertani, and D. Petrali, "Applicative system level test introduction to increase confidence on screening quality," in *DDECS*. IEEE, 2020, pp. 1–6.

[43] L. D. Rojas, K. Hess, and C. Carter-Brown, "Effectively using machine learning to expedite system level test failure debug," in *ITC*, 2019.