2023 IEEE 29th International Symposium on On-Line Testing and Robust System Design (IOLTS) | 979-8-3503-4135-5/23/$31.00 ©2023 IEEE | DOI: 10.1109/IOLTS59296.2023.10224862

Towards dependable RISC-V cores for edge computing devices

Pegdwende Romaric Nikiema[1], Alessandro Palumbo[2], Allan Aasma[3], Luca Cassano[2], Angeliki Kritikakou[1,5], Ari Kulmala[3], Jari Lukkarila[3], Marco Ottavi[4], Rafail Psiakis[3], Marcello Traiola[1]
[1]Univ Rennes, Inria, IRISA, CNRS, France, [2]Politecnico di Milano, Italy,
[3]Technology Innovation Institute, United Arab Emirates,
[4]Twente University, the Netherlands and University of Rome Tor Vergata, Italy
[5] Institut universitaire de France (IUF)
[1]{pegdwende.nikiema; angeliki.kritikakou; marcello.traiola}@inria.fr, [2]first_name.last_name@polimi.it,
[3]first_name.last_name@tii.ae, [4]m.ottavi@utwente.nl

***Abstract*—The migration of the computation from the cloud into edge devices, i.e., Internet-of-Things (IoTs) devices, reduces the latency and the quantity of data flowing into the network. With the emerging open-source and customizable RISC-V Instruction Set Architecture (ISA), cores based on such ISA are promising candidates for several application domains within the IoT family, such as automotive, Unnamed Aerial Vehicles (UAVs), industrial automation, healthcare, agriculture etc., where power consumption, real-time execution, security and reliability are of highest importance. In this emerging new era of connected RISC-V IoT devices, mechanisms are needed for a reliable and secure execution, still meeting area, energy consumption and computation time constraints of edge devices. We propose three mechanisms towards this goal, i.e., (i) a Root of Trust module for post-quantum secure boot, (ii) hardware checkers against hardware trojan horses and microarchitectural side-channel attacks, and (iii) a fine-grained dual core lockstep mechanism for real-time error detection and correction. The paper illustrates the proposed mechanisms with related motivations and implications, as well as a discussion on future research directions.**

***Index Terms*—RISC-V, OpenTitan, RoT, Secure Boot, Post-Quantum Cryptography, PQC, Security, Safety, CRYSTALS-Dilithium, Lockstep Dual Core, Recovery, Security Checker, Hardware Trojan Horses, Microarchitectural Side-Channel Attacks.**

I. Introduction

In order to reduce not only the latency, but also the quantity of data flowing into the network, there is an incentive to migrate computation from the cloud into the edge devices, i.e., Internet-of-Things (IoTs) devices. With the emerging open-source and customizable RISC-V Instruction Set Architecture (ISA), RISC-V came into the foreground gaining more and more attention of the industry [1]. RISC-V is a promising candidate for several application domains within the IoT family, such as automotive, Unnamed Aerial Vehicles (UAVs), industrial automation, healthcare, agriculture etc., where power consumption, real-time execution, security and reliability are of utmost importance [2], [3]. However, the majority of RISC-V works mainly focus on design for performance and power consumption, often neglecting dependability and security issues [4]. In this emerging new era of connected RISC-V IoT devices, mechanisms are needed for reliable and secure execution [5], meeting area, energy consumption and computation time constraints of edge devices.

To achieve this goal, such mechanisms should address the complete system stack. The fundamental mechanism, upon which the whole reliable and safety stack is built, is the secure boot, where the executed code is authenticated and its integrity is verified. After the system boot, the system execution has to be protected from faults injected either maliciously or unintended. Such faults can impact the fetching of the instructions and data from the memory to the processors or even the processor execution.

In this work, we propose three mechanisms applied at different layers of the system stack in order to achieve dependable and secure RISC-V cores. As a first step, a strong dependability foundation should be established through a secure boot scheme, bound to the hardware using a Root of Trust (RoT) module, such as OpenTitan, that uses digital signatures based on Public Key Cryptography (PKC) algorithms, such as RSA and ECC. However, the emergence of quantum computing technology makes integer factorization and discrete logarithms-based cryptography, such as RSA and ECC, respectively breakable [6]. Hence, there is a need for data, code integrity and authentication solutions that fit the post-quantum era. However, OpenTitan is unable to support existing public-key Post-Quantum Cryptographic (PQC) algorithms in its secure boot flow without major modifications, due to its resource limitations. To tackle this limitation, we propose an enhanced secure boot flow for OpenTitan RoT, modifying the OpenTitan hardware and software architecture, enabling post-quantum security.

As a second step, in order to protect from attacks during the transfer of the data from memory to the processors, mechanisms are proposed to protect from executing unwanted software and accessing illegal memory locations. To achieve that, we propose hardware mechanisms to monitor the fetching activity, the processed data and the status of the microprocessor in order to detect potentially suspicious activities, i.e., the activation of a Hardware Trojan Horse or the attempt of running a Microarchitectural Side-Channel Attack.

Last, in order to deal with hardware faults of different nature (i.e., related to manufacturing imperfections, to aging, or radiation-induced) occurring inside the processors, mechanisms typically based on redundancy are used: the same set of operations with same inputs is executed on different processors. Such approach is very effective, since the probability of having the same fault concurrently occurring on all processors is very small [7]. However, it comes in high error detection and correction cost, when it is applied at a coarse-grained level. Therefore, we propose a fine-grained lockstep version a RISC-V processor with fast error detection and correction features. We designed the approach by using High-Level Synthesis (HLS). HLS makes the design process less complex, as the processor model can easily be modified, expanded and verified, compared to HDL implementations [8].

The reminder of this paper is organized as follows: Section II presents the post-quantum secure boot mechanism; Section III introduces the hardware security checkers for anti-Trojan and anti-Side-Channel Attacks; Section IV discusses the dual-core lock step implementation for error detection and correction; finally, Section V concludes the paper.

II. Post-quantum secure boot on OpenTitan RoT

A. Motivation

OpenTitan is an open source RISC-V based Root of Trust (RoT), one use case of which is to establish the foundation of a secure chain of trust, through a secure boot procedure. As mentioned, for its secure boot, OpenTitan currently uses PKC schemes that are not post quantum safe, therefore our goal is to identify the best candidate algorithm for OpenTitan.

Current PQC mathematical solutions are based on hashes, codes, multi-variants and lattices [9]. Taking into consideration the resource constraint environments of the RISC-V edge devices, PQC secure boot algorithms should be implemented meeting area and energy consumption, as well as computation time. The first PQC hardware-based secure boot implementation using the post-quantum hash-based signature scheme is presented and compared to a hardware and software ECDSA secure boot solution [10]. PQC secure boot signatures for efficient PQC UEFI Secure Boot based on LMS and SPHINCS+ algorithms have minimal authentication time and boot time, and memory footprint [11], making these algorithms very good candidates for edge devices with restricted resources.

A survey on lattice-based cryptography implementations on constrained embedded devices, compared to traditional public-key schemes, is presented in [12]. It is shown that, as the key size increases, the performance of lattice-based cryptography schemes deteriorates. When measuring performance of different lattice-based schemes on ARM cortex-M4 microprocessor, CRYSTALS-Dilithium has the highest throughput performance in comparison to other schemes. CRYSTALS-Dilithium is also one of the two signature finalists in the third round of the NIST PQC standardization project [13], making it a good candidate to be implemented in OpenTitan. However, such a security scheme implies a complex software implementation and its introduction in the first boot stages of an IoT device is not straightforward, due to the ROM size limitations of edge devices.

To overcome this limitation and provide PQC resistant boot scheme, we extend the RTL of OpenTitan to support CRYSTALS-Dilithium acceleration. The CRYSTALS-Dilithium hardware primitive [14] is used in our design in order to verify the signature of the next stages of the boot.

B. The proposed PQC Secure Boot For OpenTitan

The proposed PQC Secure Boot on OpenTitan ensures that only payloads, which have been verified, are executed, as the system boots-up. Note that, the Secure Boot flow discussed in this manuscript excludes the later boot stages of BL0 firmware and the boot of the kernel of the supported OS. The first stage of the Boot process resides in the Boot ROM. The ROM code is programmed into the chip's ROM during manufacturing, and it is immutable. It contains a set of public keys used to verify the first boot stage (i.e. ROM_EXT) stored in flash.

The main task of the ROM code is to prepare OpenTitan for executing ROM_EXT, ensuring also that the loaded ROM_EXT is allowed to be executed on this chip (ROM Extension - Stored in flash and signed by the Silicon Creator). Figure 1 gives an overview of the two steps of the boot code. As soon as the system is powered-on/reset, the boot process

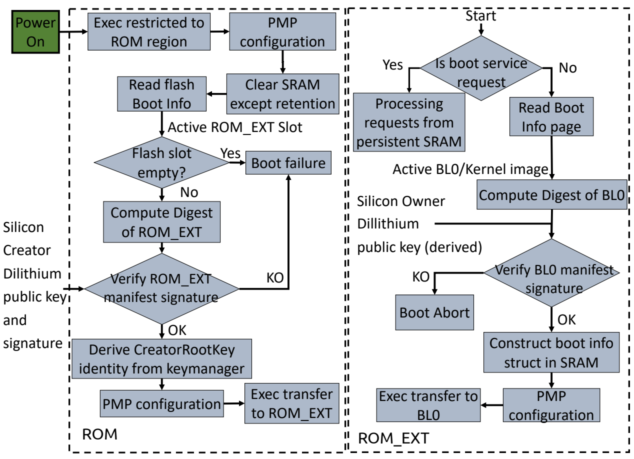
Power On
Exec restricted to ROM region
PMP configuration
Clear SRAM except retention
Read flash Boot Info
Active ROM_EXT Slot
Flash slot empty?
Yes
Boot failure
No
Compute Digest of ROM_EXT
Silicon Creator Dilithium public key and signature
Verify ROM_EXT manifest signature
KO
OK
Derive CreatorRootKey identity from keymanager
PMP configuration
Exec transfer to ROM_EXT
ROM
Start
Is boot service request
Yes
No
Processing requests from persistent SRAM
Read Boot Info page
Active BL0/Kernel image
Compute Digest of BL0
Silicon Owner Dillithium public key (derived)
Verify BL0 manifest signature
KO
Boot Abort
OK
Construct boot info struct in SRAM
PMP configuration
Exec transfer to BL0
ROM_EXT

Fig. 1. OpenTitan PQC secure boot code.

starts. The execution is restricted to the OnChip ROM, which resides inside OpenTitan. Execution of other types/areas of memory is initially prevented by the reset logic. The execution enters the ROM code and the enhanced Physical Memory Protection (ePMP of the Ibex core of OpenTitan) is enabled and configured. At this stage, only the ROM region, where the executable boot code resides, is accessible to the core. All SRAM, except for retention SRAM, is cleared. The active boot slot is loaded from the flash Boot Info. Flash boot info resides in the flash default bank. Starting with the Active ROM_EXT Slot, the ROM code performs the following steps:

- It determines if the Active ROM_EXT slot is empty, by testing for presence of the header magic value. If it is present, it continues with the validation of ROM_EXT. If it is not present, it executes the boot failure logic.
- If the Active ROM_EXT slot is empty, the boot process reads the ROM_EXT code and computes its SHA2-256 digest.
- Using CRYSTALS-Dillithium hardware primitive we perform a verification of the signature inside the ROM_EXT manifest, using the digest and the Silicon Creator Public Key, which is stored into the ROM. If the validation is successful, the process continues. If it fails, the process executes the boot failure logic.
- Afterwards, it performs system state measurements and it derives the CreatorRootKey Identity, which will reside in the key manager, as an intermediate state.
- Next, it enables the execution of the ROM_EXT by configuring the appropriate PMP entry.
- Finally, it transfers the execution to the entry point specified in the ROM_EXT manifest for the active slot.
- The execution jumps into the flash, which is eXecute In Place (XIP), and it enters in ROM_EXT code part.
- The ROM_EXT reads the Boot Info page to determine which BL0/kernel image it should be booting.
- The ROM_EXT reads the BL0 and computes its SHA2-256 digest.
- Using CRYSTALS-Dillithium hardware primitive we perform a verification of the signature inside the BL0 manifest, using the digest and the Silicon Owner Public Key derived from the Key Manager (KM) derive function. If the validation is successful, the process continues. If it fails, the process executes the boot failure logic.

Execution gets transferred to the entry point (from the BL0/Kernel image manifest) of the Silicon Owner Code. Execution enters Silicon Owner Code (i.e. BL0, Kernel Apps and Data which are all out of scope of this manuscript).

TABLE I
CRYSTALS-DILLITHIUM FPGA IMPLEMENTATION IN OPENTITAN

Design	LUTs	Relative Footprint	Clk(MHz)
OpenTitan	166.180	-	10
Dilithium	55.123	33%	2.5

We integrated the proposed approach on the original OpenTitan pipeline and evaluated on a Xilinx Virtex UltraScale+ VCU118 FPGA board. The FPGA implementation results are shown in the Table I. The proposed Dillithium hardware primitive is 33% of the whole OpenTitan design, since this implementation is tuned for high performance. In order to avoid synchronization issues, we had to clock it down to 1/4 of OpenTitan's frequency. To reduce the above cost, as a future work, we will trade-off performance for area footprint gains in our next hardware primitive design, and implement hardware accelerator solutions to offload the lattice computations of Dilithium algorithm. Offloading only the most demanding parts of the algorithm in the hardware, i.e., the random generation and the polynomial multiplication operations, could be studied and compared against pure hardware implementation.

III. Hardware Security Checkers

A. Motivation

The RISC-V architecture natively features a number of security extensions like privilege levels, physical memory protection and cryptography [15]. Nevertheless, these native mechanisms do not protect the microprocessor from two rising menaces: *Hardware Trojan Horses* and *Microarchitectural Side-Channel Attacks* [16].

Hardware Trojan Horses (HTHs) have been considered as a purely academic issue for a long time, since they generally exposed limited complexity and dangerousness. Nevertheless, a new menace recently raised, that made HTH a concern also for industries: the *software exploitable HTHs* [17]. Complex and highly dangerous HTHs may infect IP cores implementing microprocessors: such HTHs may allow the attacker to execute malicious software, to modify the running software, to acquire unauthorized privileges or to steal secret information [18]. A recent real-world HTH, the *Rosenbridge* backdoor, has been found in a commercial Via Technologies C3 processor [19]. This HTH can be activated and exploited via software to enter the supervisor mode of the system[1]. Although several circuit-level design-time HTH detection techniques have been proposed [20]–[22], there is a growing interest in system-level techniques that allow to obtain a trusted system built with untrusted components [23], [24] or a trusted software execution over a (partially) untrusted system [25], [26]. We argue that existing solutions do not take into account those HTHs that change the functionality of the system by making the CPU run normal (but unexpected) instructions without changing privilege mode. In other words, none of these works checks whether the microprocessor is executing an unwanted software and whether it is accessing illegal memory locations.

Another menace that raised in the very last years are *Microarchitectural Side-Channel Attacks (MSCAs)* [27], e.g., Spectre and Meltdown. These attacks allow to steal unauthorized information without requiring the attacker to have physical access to the system under attack. Indeed, such attacks only rely on the exploitation of *legal* HW features, e.g., speculative execution and branch prediction, and on the observation of the timing behavior of the system while running sensitive applications. Several countermeasures against MSCAs have been proposed in the last few years [28]. *Constant-time* techniques, rely on making execution time constant to protect secret information; compile-time techniques have been proposed to identify and modify those instruction sequences that may open the door to MSCAs; and OS-level solutions based on cache partitioning, and periodic cache flushing have been proposed. All these techniques suffer from a limited applicability and a significant slowdown. Finally, a large

[1]After the publication of [19] Via Technologies officially commented that this behavior was due to an undocumented feature meant for debug.

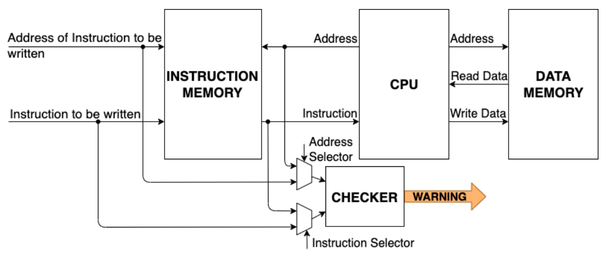

Fig. 2. The security architecture.

number of techniques that exploit machine learning and the observation of hardware performance counters exist. These techniques achieve very high detection accuracy, but they all require either multithreading or an additional (trusted and not attacked core) dedicated to the execution of the detection engine. Therefore, these techniques can be applied to high performance computing systems, e.g., high-end servers, but they are not suitable for edge devices and low-end systems, e.g., smart cards or automotive embedded systems, where a single core is available and the operating system is either not available or extremely simple.

B. *The Proposed Hardware Security Solutions*

Our solutions (whose high-level representation is depicted in Figure 2) aim at allowing the system integrator to build a trusted system by purchasing possibly untrusted components. Therefore, we consider the IP provider as the attacker; on the other hand the system integration employees and the foundry are considered trusted. We introduce Hardware Security Checkers (HSCs) in the architecture, such that the HSCs are able to monitor the fetching activity, the processed data and the status of the microprocessor in order to detect potentially suspicious activities, i.e., HTHs and/or MSCAs activations. These HSCs work in two phases: in a first design-time *configuration* step they *learn*[2] which instructions/sequences of instructions are legitimate. In subsequent runtime *monitoring* step, they check whether the activity of the microprocessor corresponds to previously learned legitimate profiles or not.

1) Detection of Hardware Trojan Horses: In [29] we proposed a HSC based on Bloom filters to protect the microprocessor against HTHs in the main memory. The HSC is configured while the software is installed in the instruction memory by storing information about legal instructions and corresponding memory addresses. At runtime, the HSC monitors the memory locations accessed by the microprocessor and the corresponding fetched instructions and based on this information it queries the Bloom filter. In case the accessed memory address and/or the corresponding fetched instruction are not legitimate the HSC will raise an alarm. A lighter version of the HSC has been proposed in [30].

We integrated the proposed anti-HTH security checker in the RI5CY version of the PULPINO platform, which is a small 4-stage RISC-V core. When synthesized on a Xilinx Artix XC7A35T, RI5CY requires 15097 LUTs and 9881 FFs and it works at about 50MHz with a total power consumption of 127mW. We considered the following benchmarks: `Binary Search (BinS)`, `Matrix Multiplication (MM)`, `Bubble Sort (BubS)`, `Quick Sort (QS)`, `Sudoku Solver (SS)` and `Motion Detection (MD)`. We simulated 10,000 cases, where the HTH forces a malicious fetch from an unexpected memory address and 10,000 cases where the HTH modifies the fetched instruction read from a legitimate address. Table II reports false positive (false alarms) and false negative (undetected attacks) rates for the two attack scenarios. It can be observed that we always have 0% false alarm rate and on average only about 2% attacks are not detected, but only when the HTH modifies the fetched instruction. Regarding the overhead, we introduce about 10% area increase, about 2% power consumption increase and no working frequency reduction.

TABLE II
HTH DETECTION ACCURACY

Bench.	Adddress modification		Instruction modification	
	FP	FN	FP	FN
`BinS`	0%	0%	0%	2.25%
`MM`	0%	0%	0%	0.40%
`BubS`	0%	0%	0%	3.01%
`QS`	0%	0%	0%	3.91%
`SD`	0%	0%	0%	0.72%
`MD`	0%	0%	0%	2.83%
AVG	0%	0%	0%	2.18%

2) Detection of Microarchitectural Side-Channel Attacks: An approach similar to the previously presented ones, but meant for the identification of the activation of MSCAs has been proposed in [31]. We exploited the Count-Min Sketch model to detect at runtime the execution of pre-defined and reproducible sets of suspicious instruction patterns that are representative of specific MSCAs. These instruction patterns, one for each attack of interest, are carefully identified by the security engineer at design and then used to configure the HSC. Therefore, our proposal is able to check several microarchitectural attacks in parallel thus representing a flexible and scalable security solution.

We integrated our HSC into the RSD core, which is a 32-bit, speculative out-of-order, super-scalar, two-fetch front-end and five-issue back-end pipelines RISC-V core with 16 KByte Instruction cache. We implemented the target microprocessor on a Virtex7 FPGA employing 18334 LUTs, 10885 FFs, 4512 LUTRAM cells and 17 BRAM cells and worked at 57 MHz with an estimated power consumption of about 0.926W. We considered `Coremark`, `Towers`, `RSort` and `Median` as benign programs under attack and `Spectre`, `Orchestration`, `Flush+Reload` and `Rowhammer` as MSCAs. We ran 10000 experiments for each benign program – attack program and the result has been that 100% of the attacks has been detected. Then, we measured the number of false alarms raised by the introduced HSC: it ranged from

[2]The term "learn" is here used although the proposed solutions do not rely on machine learning/artificial intelligence.

about 80% down to 0% depending on the configuration of the HSC itself. The smallest HSC configuration that allowed us to get both 0% false positive and false negative rates introduced about 10% area overhead, about 4% power consumption increase and no working frequency reduction.

IV. Lock-step dual core

A. Motivation

Processor redundancy can be implemented in a *non-intrusive* or in an *intrusive* way, using *homogeneous* or *heterogeneous* processors. *Non-intrusive* approaches introduce redundancy at core level without modifying the internal processor micro-architecture, thus they are typically used when internal micro-architecture details are not available or difficult to modify, e.g., Commercial Off-The-Shelf (COTS) processors. For instance, two identical MicroBlaze soft cores are used to implement a homogeneous Dual Core LockStep (DCLS) [32], where the faulty processor is excluded from computation and repaired through reconfiguration, while the correct processor keeps working. A heterogeneous non-intrusive DCLS uses a RISC-V soft core along with an ARM A9 hard core [33]. Checkpoints are used in the application to check for mismatch between the cores and a roll-back strategy is used upon detection. Heterogeneous approaches may reduce performance, with the low-speed processor being the upper bound to the DCLS performance. Moreover, performing lockstep with hard cores requires specific architecture support, which is not present on all processors [33]. Overall, non-intrusive approaches are simple to implement, as no micro-architectural modifications are needed. However, they leave to the application level the responsibility to define a correction strategy (e.g., through regular checkpoints), leading to possible high timing overhead.

On the other hand, *intrusive approaches* modify the internal processor micro-architecture, improving flexibility when implementing correction mechanisms (e.g. rollback). For instance, the Dynamic Adaptive Redundancy Architecture (DARA) is a homogeneous intrusive DCLS approach applied to RISC ISA SH-2 processors and achieves error correction through rollback [34]. DARA adds additional hardware to check the consistency of all pipeline stage registers, between the lockstep cores. However, DARA does not support faults occurring in branch instructions. Another homogeneous intrusive approach uses two virtual RISC-V cores to implement a DCLS through fine-grained interleaved multitasking to tackle common-mode failures (CMFs) [35]. Other approaches extend the pipeline registers with error detection and correction codes, e.g., Duckcore extends a RISC-V core with Single Error Correction Double Error Detection (SECDED) in the pipeline registers [36]. However, such an approach can add significant overhead due to the encode and decode time and do not guarantee protection against faults in the computation logic of pipeline stages. Other approaches triplicate components within the RISC-V core to enhance its reliability. For instance, Control and Status Registers, Program Counter and the register file [37], FFs, LUTs, BRAMS, and DSPs [38], and the arithmetic and logic unit (ALU) are triplicated [39]. Overall, intrusive approaches offer high flexibility to implement efficient fault correction mechanisms but can be challenging to realize. Indeed, existing intrusive approaches are based on HDL implementations, which are usually quite complex.

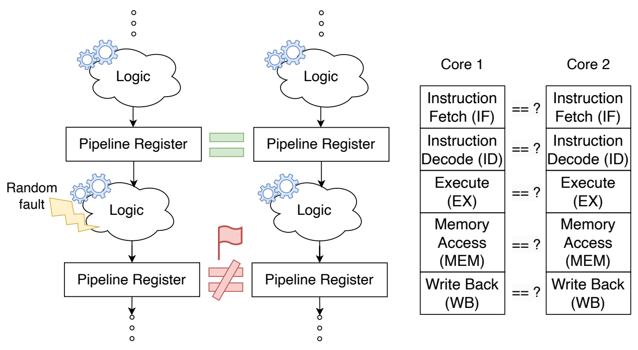

Fig. 3. Dual Core Lock-Step principle illustration

Finally, the above described approaches do not focus on providing bounded error detection and correction time. However, this is key when embedded systems are employed in domains requiring both hard real-time and reliable execution, where the Worst-Case Execution Time (WCET) of the application has to be bounded also in presence of possible faults.

B. Proposed approach

To deal with the aforementioned problem, we proposed an intrusive homogeneous RISC-V-based dual-core lock step implementation providing transient error detection and correction with a bounded number of extra clock cycles.

In particular, in [40] we resorted to the open-source Comet RV32I RISC-V architecture[3] [8] to implement two intrusive lockstep designs, i.e., *Partial Shadow Register with Rollback (PSRR)* and *Full Shadow Register (FSR)*, through High-Level Synthesis (HLS). We used two identical RISC-V cores executing the same instructions at each clock cycle. Each pipeline stage stores the result of its logic computation in a pipeline register. At each cycle, the proposed mechanism checks for execution consistency by comparing the pipeline registers of the two cores, as sketched in Figure 3. If no error is detected, the execution runs normally. Otherwise, the detected error indicates that a fault impacted the logic, which in turn generated a wrong result, or that a fault impacted the pipeline register itself, flipping a bit in the result. In this case, immediate correction is applied. In PSRR approach, the faulty instruction is re-fetched and goes through the whole pipeline again, similar to DARA [34]. Previous instructions still in the pipeline are left untouched as their execution is not impacted by the error; however, subsequent instructions already in the pipeline are discarded. In FSR approach, when no faults are detected, we create a backup copy of all the pipeline registers. Then, upon fault detection, results of the current computation are discarded, the pipeline registers of both cores are restored

[3]https://gitlab.inria.fr/srokicki/Comet/-/tree/master

with the backup values, and finally both cores re-execute the cycle that was impacted by a fault.

Moreover, in [41] we expose the following key aspect: transient faults affecting the cores impacts not only the functional behavior of an application, but it also has a significant impact on its timing behavior, affecting WCET estimations. To achieve that, we leverage typical fault-free WCET estimations to be fault-aware, by taking into account the impact of transient faults occurring on cores. More precisely, we firstly perform a vulnerability analysis on a target system through extensive fault injection. The analysis verifies not only functional correctness, but also timing correctness of applications, when executed on a core. Then, we apply a typical measurement-based WCET estimation method to verify the impact of faults on WCET estimation.

We experimented on benchmarks from TACLeBench (`Binary Search` and `Prime`), MiBench (`Qsort`), AxBench (`Moving Average`), and Polybench (`Matrix Multiplication`). From the obtained results, we observe that the application execution time can be significantly increased under the presence of transient faults, up to 700%, compared to the application execution time without faults. Furthermore, the distribution of execution time traces is significantly modified, compared to the fault-free distribution. The above observations have direct consequences; the time required to finish execution under faults can be significantly higher than the fault-free WCET. Thus, existing approaches should use watchdog timers, in order to bound the impact of transient faults on the application execution time, and keep safe the overall schedule. When the timer expires or an error is detected, the application requires to be re-executed, fully or partially, depending on the approach, leading to high error detection and correction timing overhead. We demonstrated that the proposed DCLS approach corrects faults as soon as they occur – before being propagated and affecting the execution time. Thus, the approach provides bounded timing overhead (i.e., two clock cycles in the FSR case) and restores WCET estimations.

V. Discussion and conclusion

Edge devices based on RISC-V architectures are emerging, leading to the need for secure and reliable systems. The current work presents three mechanisms deployed on RISC-V devices towards this goal. As signature verification algorithms are proven to be unsafe in the post-quantum era, we proposed a post-quantum safe secure boot mechanism for IoT devices using OpenTitan RoT. Moreover, in order to detect potentially suspicious activities, we show hardware mechanisms to monitor the fetching activity, the processed data and the status of the microprocessor. Finally, two mechanisms with bounded WCET overhead have been shown on a dual-core lockstep intrusive RISC-V to detect and recover from radiation-induced transient faults.

Moreover, we would like to emphasize the advantages that the RISC-V open ISA can bring to the dependability research community. The large availability of open-source core implementations and the large community around RISC-V enable researchers to easily implement and test their solutions addressing different dependability issues. However, we notice that there are not many approaches trying to address different challenges with a unified approach, e.g., the same hardware used to provide both fault tolerance and confidentiality [42]. Rather, experts of different domains (e.g., reliability and security) usually decide independently about their requirements of interest, often with limited interactions. Moreover, non-functional requirements are often considered as a optional commodity, which increases the cost of the related solutions. We argue that researchers from different groups with different expertise should federate and push dependability requirements to be among the main design requirements for future RISC-V cores to be used in mainstream applications. In this way, unified highly-optimized implementations ensuring efficient and dependable operations can thrive and power the RISC-V cores for edge computing devices of tomorrow.

Acknowledgment

Part of this work has been funded by the French National Research Agency (ANR) through the FASY research project (ANR-21-CE25-0008). Part of this work has been funded by the Technology Innovation Institute in Abu Dhabi, UAE.

References

[1] C. Palmiero *et al.*, "Design and implementation of a dynamic information flow tracking architecture to secure a risc-v core for iot applications," in *2018 IEEE High Performance extreme Computing Conference (HPEC)*. IEEE, 2018, pp. 1–7.

[2] J. Abella *et al.*, "Security, reliability and test aspects of the risc-v ecosystem," in *2021 IEEE European Test Symposium (ETS)*, 2021, pp. 1–10.

[3] J. Anders *et al.*, "A survey of recent developments in testability, safety and security of risc-v processors," in *2023 28th IEEE European Test Symposium (ETS)*, 2023.

[4] A. Dörflinger *et al.*, "A comparative survey of open-source application-class risc-v processor implementations," in *Proceedings of the 18th ACM International Conference on Computing Frontiers*, ser. CF '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 12–20. [Online]. Available: https://doi.org/10.1145/3457388.3458657

[5] Y. H. Hwang, "Iot security & privacy: threats and challenges," in *Proceedings of the 1st ACM workshop on IoT privacy, trust, and security*, 2015, pp. 1–1.

[6] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM review*, vol. 41, no. 2, pp. 303–332, 1999.

[7] M. Cui *et al.*, "Fault-tolerant mapping of real-time parallel applications under multiple dvfs schemes," in *IEEE RTAS*, 2021, pp. 387–399.

[8] S. Rokicki *et al.*, "What you simulate is what you synthesize: Designing a processor core from c++ specifications," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019, pp. 1–8.

[9] V. Mavroeidis *et al.*, "The impact of quantum computing on present cryptography," *arXiv preprint arXiv:1804.00200*, 2018.

[10] V. B. Kumar *et al.*, "Post-quantum secure boot," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2020, pp. 1582–1585.

[11] P. Kampanakis *et al.*, "Post-quantum lms and sphincs+ hash-based signatures for uefi secure boot," *Cryptology ePrint Archive*, 2021.

[12] A. Khalid *et al.*, "Lattice-based cryptography for iot in a quantum world: Are we ready?" in *2019 IEEE 8th international workshop on advances in sensors and interfaces (IWASI)*. IEEE, 2019, pp. 194–199.

[13] J. Zheng *et al.*, “Parallel small polynomial multiplication for dilithium: A faster design and implementation,” in *Proceedings of the 38th Annual Computer Security Applications Conference*, ser. ACSAC ’22. New York, NY, USA: Association for Computing Machinery, 2022, p. 304–317. [Online]. Available: https://doi.org/10.1145/3564625.3564629

[14] L. Beckwith *et al.*, “High-performance hardware implementation of crystals-dilithium,” in *2021 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 2021, pp. 1–10.

[15] A. Waterman *et al.*, “The risc-v instruction set manual volume ii: Privileged architecture document version 20190608-priv-msu-ratified,” RISC-V Foundation, Tech. Rep., 2019.

[16] L. Cassano *et al.*, “Is risc-v ready for space? a security perspective,” in *2022 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2022, pp. 1–6.

[17] X. Wang *et al.*, “Software exploitable hardware trojans in embedded processor,” in *2012 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2012, pp. 55–58.

[18] Y. Jin *et al.*, “Exposing vulnerabilities of untrusted computing platforms,” in *Proc. Int. Conf. Computer Design*, 2012, pp. 131–134.

[19] C. Domas, “Hardware backdoors in x86 cpus,” https://i.blackhat.com/us-18/Thu-August-9/us-18-Domas-God-Mode-Unlocked-Hardware-Backdoors-In-x86-CPUs-wp.pdf, 2018.

[20] X. Chuan *et al.*, “An efficient triggering method of hardware Trojan in AES cryptographic circuit,” in *Proc. Int. Conf. Integrated Circuits and Microsystems*, 2017, pp. 91–95.

[21] Y. Liu *et al.*, “Scca: Side-channel correlation analysis for detecting hardware trojan,” in *Proc. Int. Conf. Anti-counterfeiting, Security, and Identification*, 2017, pp. 196–200.

[22] A. Palumbo *et al.*, “Is your fpga bitstream hardware trojan-free? machine learning can provide an answer,” *Journal of Systems Architecture*, vol. 128, p. 102543, 2022.

[23] J. Dubeuf *et al.*, “Run-time detection of hardware trojans: The processor protection unit,” in *2013 18th IEEE European Test Symposium (ETS)*, 2013, pp. 1–6.

[24] G. Bloom *et al.*, “Os support for detecting trojan circuit attacks,” in *2009 IEEE International Workshop on Hardware-Oriented Security and Trust*, 2009, pp. 100–103.

[25] L. Cassano *et al.*, “Deton: Defeating hardware trojan horses in microprocessors through software obfuscation,” *Journal of Systems Architecture*, vol. 129, p. 102592, 2022.

[26] ——, “On the optimization of software obfuscation against hardware trojans in microprocessors,” in *2022 25th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, 2022, pp. 172–177.

[27] A. P. Fournaris *et al.*, “Exploiting hardware vulnerabilities to attack embedded system devices: a survey of potent microarchitectural attacks,” *Electronics*, vol. 6, no. 3, p. 52, 2017.

[28] Y. Lyu *et al.*, “A survey of side-channel attacks on caches and countermeasures,” *Journal of Hardware and Systems Security*, vol. 2, no. 1, pp. 33–50, 2018.

[29] A. Bolat *et al.*, “A microprocessor protection architecture against hardware trojans in memories,” in *2020 15th Design & Technology of Integrated Systems in Nanoscale Era (DTIS)*. IEEE, 2020, pp. 1–6.

[30] A. Palumbo *et al.*, “A lightweight security checking module to protect microprocessors against hardware trojan horses,” in *2021 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. IEEE, 2021, pp. 1–6.

[31] K. Arıkan *et al.*, “Processor security: Detecting microarchitectural attacks via count-min sketches,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 30, no. 7, pp. 938–951, 2022.

[32] A. Hanafi *et al.*, “Dual-lockstep microblaze-based embedded system for error detection and recovery with reconfiguration technique,” in *2015 Third World Conference on Complex Systems (WCCS)*, 2015, pp. 1–6.

[33] A. B. De Oliveira *et al.*, “Lockstep Dual-Core ARM A9: Implementation and Resilience Analysis Under Heavy Ion-Induced Soft Errors,” *IEEE Transactions on Nuclear Science*, vol. 65, no. 8, pp. 1783–1790, Aug. 2018. [Online]. Available: https://ieeexplore.ieee.org/document/8401918/

[34] J. Yao *et al.*, “Dara: A low-cost reliable architecture based on unhardened devices and its case study of radiation stress test,” *IEEE Transactions on Nuclear Science*, vol. 59, no. 6, pp. 2852–2858, 2012.

[35] M. T. Sim *et al.*, “A dual lockstep processor system-on-a-chip for fast error recovery in safety-critical applications,” in *IEEE IECON*, 2020, pp. 2231–2238.

[36] J. Li *et al.*, “Duckcore: A fault-tolerant processor core architecture based on the risc-v isa,” *Electronics*, vol. 11, no. 1, 2022.

[37] L. Blasi *et al.*, “A RISC-V fault-tolerant microcontroller core architecture based on a hardware thread full/partial protection and a thread-controlled watch-dog timer,” in *APPLEPIES*, 2019, pp. 505–511.

[38] A. E. Wilson *et al.*, “Neutron radiation testing of fault tolerant risc-v soft processor on xilinx sram-based fpgas,” in *IEEE SCC*, 2019, pp. 25–32.

[39] D. A. Santos *et al.*, “A low-cost fault-tolerant risc-v processor for space systems,” in *DTIS*, 2020, pp. 1–5.

[40] P. R. Nikiema *et al.*, “Design with low complexity fine-grained dual core lock-step (dcls) risc-v processors,” in *2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks - Supplemental Volume (DSN-S)*, July 2023.

[41] ——, “Impact of transient faults on timing behavior and mitigation with near-zero wcet overhead,” in *ECRTS 2023 - 35th Euromicro Conference on Real-Time Systems*, Vienna, Austria, July 2023.

[42] N. I. Deligiannis *et al.*, “Towards the integration of reliability and security mechanisms to enhance the fault resilience of neural networks,” *IEEE Access*, vol. 9, pp. 155 998–156 012, 2021.