Multi-Partner Project: Advancing the EDA Tools Landscape for the European RISC-V Ecosystem in TRISTAN

Fatma Jebali*, Caaliph Andriamisaina*, Mathieu Jan*, Wolfgang Ecker†, Florian Egert‡, Bernhard Fischer‡, Alessio Burrello§, Daniele Jahier Pagliari§, Sara Vinco§, Giuseppe Tagliavini¶, Ingo Feldner‖, Andreas Mauderer‖, Axel Sauer‖, Arnór Kristmundsson**, Alexander Schober**, Téo Bernier††, Matti Käyrä‡‡, Ulf Schlichtmann[x], Rocco Jonack[xi]

* Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France; † Infineon Technologies AG;
‡ Siemens Foundational Technology, Vienna, Austria; § Politecnico di Torino, Italy;
¶ University of Bologna, Italy; ‖ Robert Bosch GmbH, Germany;
** Codasip GmbH, Germany; †† Thales Research & Technology, France;
‡‡ Tampere University, Finland; [x] Technical University of Munich, Germany;
[xi] MINRES Technologies GmbH, Germany

The TRISTAN project aims to expand and industrialize the European RISC-V ecosystem to compete effectively with existing commercial alternatives. This initiative specifically targets the critical challenges in the development of Electronic Design Automation (EDA) tools, essential for RISC-V-based solutions, by leveraging the synergy between the open-source community and industrial solutions. This paper presents an overview of the current landscape of TRISTAN's EDA flow, highlighting specific tools and methodologies that streamline the early design phases of RISC-V-based systems. We explore the unique features of these tools, emphasizing how they complement each other to strengthen the overall design process.

***Index Terms*—TRISTAN, RISC-V, Electronic Design Automation**

I. Introduction

The European Chips Act considers the creation and expansion of a RISC-V open-source ecosystem to be a strategic investment towards Europe's ambition to double the value of the design and production of semiconductors in Europe by 2030 [1]. In this context, the TRISTAN project has been funded under the Chips JU program and comprises 46 stakeholders, with the goal of consolidating the European RISC-V ecosystem to compete with existing proprietary alternatives and to propose open-source solutions [2]. The *Electronic Design Automation* (EDA) team consists of around twenty partners from both industry and academia, combining their expertise to advance RISC-V tooling in support of the European roadmap [3].

Central to the TRISTAN initiative is a comprehensive suite of EDA tools and methods designed to accelerate and streamline the development of RISC-V-based systems. This paper showcases a range of the proposed tools intended to confront the following key requirements:

- Open-source tools for pre-silicon validation and early architectural exploration, with performance and power considerations alongside SW validation, fostering collaborative tool development.
- Automated flows for model generation, including top-down high-level synthesis and bottom-up ML-based model generation, essential for making HW design accessible to a broader audience without requiring in-depth expertise.
- Interoperability frameworks and open-source standards for interfacing between open-source and proprietary tools, ensuring flexible design flows and compatibility across different IP models.
- Verification tools for security and safety assessment of HW design, suitable for critical applications such as automotive and space.

The TRISTAN EDA contributions focus mainly on the early front-end design phases, including architectural exploration, design and implementation, and verification. With the project being in an intermediate stage, this paper captures the current tool landscape and highlights a selection of the proposed tools and methods for each of the aforementioned design phases.

II. Overview of TRISTAN EDA tools

The TRISTAN EDA tools aim to enrich the RISC-V ecosystem and streamline the development of Integrated Circuits (ICs). Figure 1 categorizes these tools based on their open-source or commercial nature and their correspondence to the four main stages of a typical IC design flow, i.e., architecture, design and implementation, verification, synthesis and layout.

The architecture stage involves high-level and early design tasks, including architectural exploration and behavioral design of core and custom extension IPs. High-level models, instruction set simulators (ISS), and virtual prototyping solutions enable the estimation of extra-functional system properties, such as performance or power, as well as the validation of HW/SW functionality. These models and simulation tools are

978-3-9826741-0-0/DATE25/© 2025 EDAA

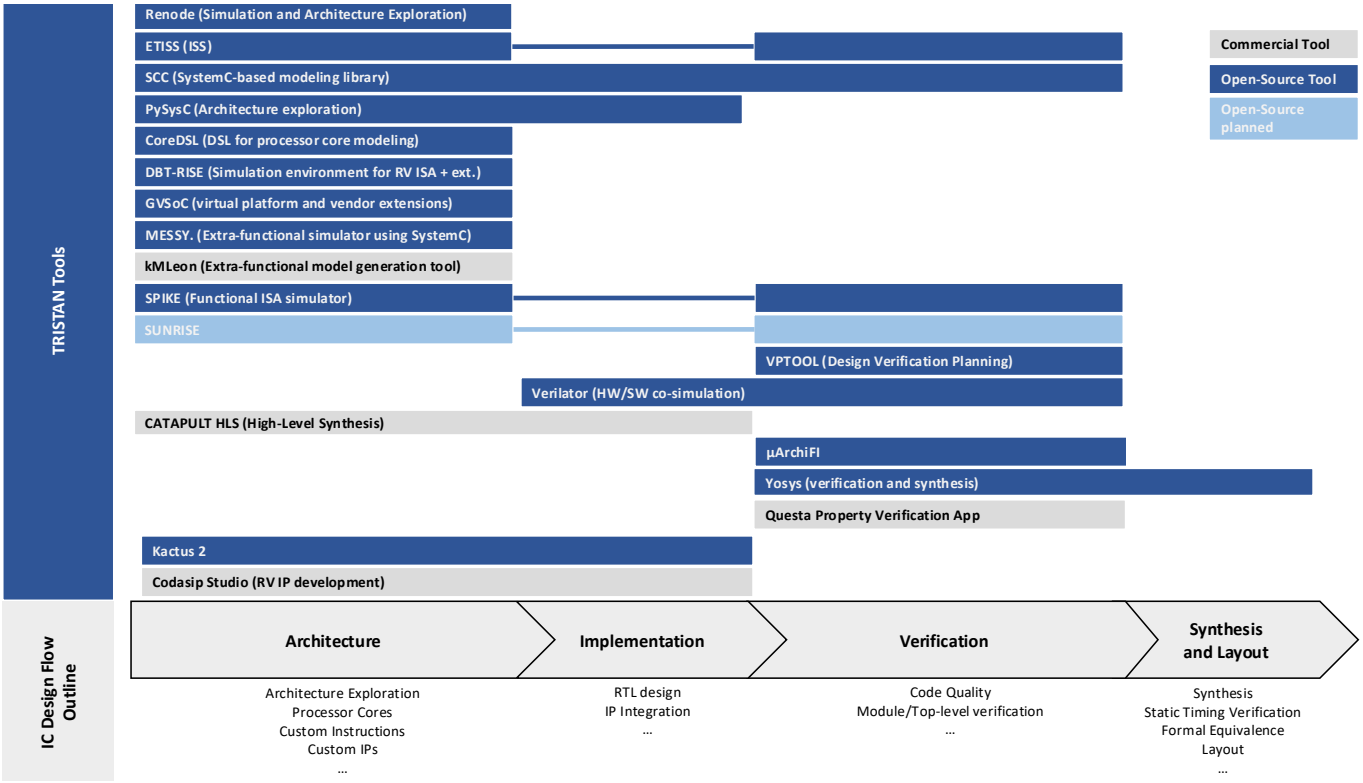

Fig. 1. Overview of the current TRISTAN tools and their IC Design Flow categorization. Grey- or blue-colored blocks correspond to the respective open-source or commercial tools. The tools are classified based on their association with the respective phases of the IC design flow.

crucial for exploring the extensive design space formed by the various configurations of cores, ISA extensions, and peripheral IP within the RISC-V ecosystem.

The implementation stage focuses on tools to streamline the design, implementation, and integration of RTL IPs. TRISTAN tools in this category cover automated HDL generation and facilitate standardized IP reuse and integration, enhancing efficiency throughout all phases of IC development.

The verification stage encompasses solutions for simulation-based and formal verification methods on both component and system levels. These capabilities are essential for improving code quality and robustness, as well as verifying safety and security properties.

In addition to tools specific to each design stage, some TRISTAN tools and frameworks cover multiple stages allowing different abstraction levels to be mixed, in a co-simulation environment or through specific APIs and DSLs (Domain Specific Languages). This is particularly useful for assessing consistency between reference models and RTL implementations, and for mapping SW requirements to different HW abstraction levels to predict PPA metrics (i.e. Performance, Power, Area) for HW design exploration.

The last IC design stages are out of the scope of TRISTAN. Because of the limited availability of open-source approaches at the synthesis and layout stage, existing commercial tools and methodologies are used for these critical and complex steps.

The following sections present a subset of TRISTAN tools and methodologies for each of the early three design stages outlined above.

III. Tools and Methods for Architecture Exploration

In the early stages of HW design, high-level models play a critical role in SW validation and design space exploration, allowing functionality, performance, and power to be evaluated without the need for physical prototypes. High simulation speed is essential to support the iterative nature of HW/SW co-design, enabling rapid design refinement and exploration. Balancing accuracy and speed is crucial to maximize efficiency, enabling thorough performance optimization and early power analysis within feasible time frames.

Given the wide range of tools and methods available for architectural exploration, this section focuses on three critical categories. First, we introduce GVSoC, a virtual prototyping tool of RISC-V platforms. Next, we present MESSY and kMLeon, which are tailored for modeling extra-functional properties, including performance and power. Finally, we introduce SUNRISE, a comprehensive framework for evaluating SoC designs using different simulation solutions. Together, these tools give HW designers critical insight to refine designs and improve overall system performance.

A. GVSoC for RISC-V Virtual Prototyping

GVSoC [4] is an event-driven simulator for modeling RISC-V-based platforms, including cores, HW units, and peripherals. This tool is specifically designed for scenarios where a balance

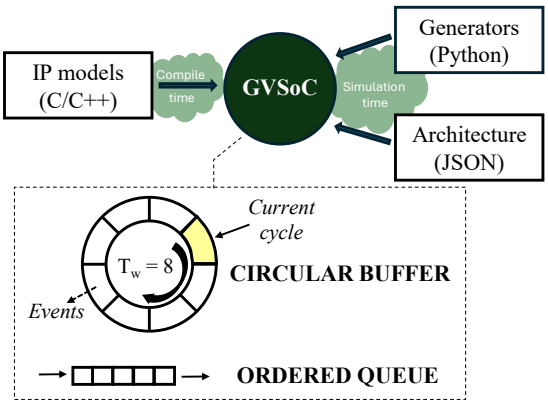

Fig. 2. Overview of GVSoC. The diagram depicts the main components of the simulation tool (C/C++ IP models, Python generators, JSON architectural description) and provides an overview of the internal timing model.

between accuracy and simulation speed is critical. Unlike functional simulators, which sacrifice accuracy for execution speed, GVSoC can model complex architectural details at different levels without the strict need to fully emulate the cycle-by-cycle operations. This feature facilitates virtual prototyping and design space exploration (DSE) tasks, where achieving realistic workload simulations and I/O interactions limiting the simulation time is essential. Experimental assessments show that GVSoC achieves a peak simulation speed of 25 MIPS (Million Instructions Per Second) and a maximum of 10% mismatch between simulation and cycle-accurate emulation approaches (i.e., using RTL or FPGA).

The GVSoC design, depicted in Figure 2, combines Python and C++ to achieve flexibility in the configuration phase and performance during execution. Python scripts describe the platform configuration, enabling fast setup and parameter adjustment, while C++ classes provide efficient, detailed models for system components, ensuring high simulation speed. Designers can further reduce the programmatic complexity of the architectural configuration through JSON files. This multi-language approach significantly simplifies the support of new ISA extensions. In TRISTAN, we extended the core model to support mixed-precision arithmetic instructions and model new peripherals (i.e., a smart DMA engine and a matrix multiplication accelerator). Architecture designers implemented the support for ISA extensions and HW units by extending foundational C++ classes, while system designers leveraged these models to configure new platform targets.

GVSoC includes a *clock engine* that simulates a clock source with a queue of associated events, each carrying a data payload and a callback function. Events scheduled together are grouped within a time window T_w in a circular buffer, allowing cycle-by-cycle execution, with simultaneous events processed sequentially. Any instance of a model component can add new events anytime if they fall within the T_w window; otherwise, they are stored in an ordered queue and processed once a circular lap completes. The choice of the algorithm to compute the slot and the position in the ordered queue represents the main trade-off between accuracy and simulation performance when a designer provides a new model.

B. Methodologies for Extra-Functional Property Modeling

Considering extra-functional properties, such as power and temperature, is crucial to tame the complexity of modern heterogeneous systems. Two methods are available in TRISTAN for modeling extra-functional properties that are compatible with integration into virtual prototyping tools: (1) kMLeon uses machine learning to automatically generate models, eliminating the need for in-depth hardware knowledge and enabling efficient high-level model generation from lower-level descriptions; (2) MESSY is based on SystemC modeling that can be tailored to specific user requirements without the need for lower-level models, making it ideal for early-stage design exploration.

1) kMLeon for ML-based Performance Modeling: The availability of cycle-accurate models and simulators enables automatic derivation of more abstract, higher-level models. These abstractions speed up evaluation processes such as SW performance evaluation and HW architecture exploration.

In TRISTAN, we focus on the automated generation of performance models from RTL simulations using machine learning techniques. These models are designed to seamlessly integrate with high speed simulators, including instruction set simulators and virtual prototypes, to achieve an optimal balance between simulation speed and accuracy. The resulting flow is depicted in Figure 3.

For model generation, we use kMLeon [5], an ML-based tool that automates the generation of extra-functional properties. It provides full automation for model exploration and validation, allowing users to select models that best meet their needs in terms of accuracy, speed, and interpretability.

For model integration with fast simulators, the only requirement is the support for instruction-level counting. The framework operates in two phases: an online phase where the simulator gathers per-type instruction counts and an offline phase where these counts are fed into the ML-based model to predict performance, thereby enhancing both accuracy and simulation efficiency.

This methodology has been applied to the OpenHWGroup's CVA6 architecture [6], available in TRISTAN. Using Verilator,

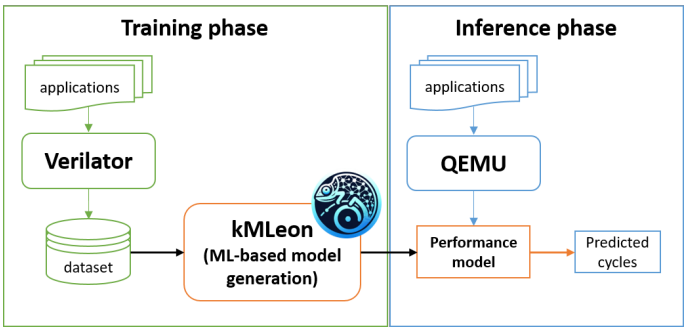

Fig. 3. Overview of the ML-driven model generation framework. The training phase uses cycle-accurate traces to generate performance models, which then predict new applications performance in the inference phase.

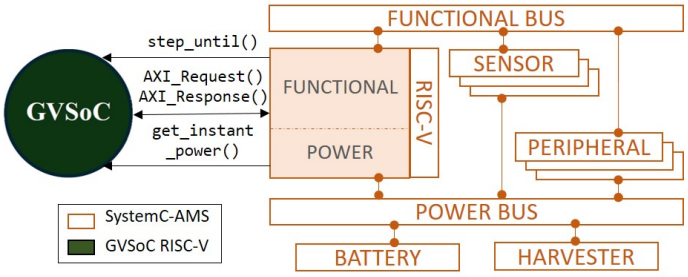

Fig. 4. MESSY: integration of SystemC-AMS and GVSoC through a bus-based architecture to achieve extra-functional awareness in virtual platforms.

detailed RTL traces are generated, formatted, and then provided as input to kMLon for model generation and exploration. QEMU [7] has been used as a fast simulator that supports modern architectures and conforms to the latest RISC-V community standards, ensuring compatibility with both standard and custom extensions. First experimental results show that, with minimal impact on overall simulation speed compared to QEMU, the proposed framework achieves an average relative error in performance estimation of less than 7% and a maximum relative error of less than 19% compared to Verilator.

2) MESSY for SystemC Power Modeling: MESSY (**M**ulti-layer **E**xtra-functional **S**imulator in **S**ystemC) is an open-source framework that integrates functional RISC-V simulation (achieved with GVSoC) with SystemC-AMS, that is used to model extra-functional aspects. The choice of GVSoC allows a tight integration of TRISTAN RISC-V tools (Section III-A), while SystemC AMS [8] is a widespread solution for virtual platforms and for simulating extra-functional aspects [9], [10].

The proposed architecture to integrate GVSoC and SystemC AMS exploits a *bus-centric paradigm* (Figure 4). The simulation features one bus for each modeled aspect (e.g., functionality and power), and each system component may be connected to each bus: e.g., a core will have a functional implementation that describes instruction processing connected to the functional bus, and a power model that estimates the corresponding power demand exported to the power bus. The mutual interdependency between functional and extra-functional aspects is then achieved through an *information exchange between the models of the same component* (e.g., the instruction being processed by the core is propagated to the power model to estimate the corresponding current demand).

Both GVSoC and SystemC AMS are event-driven. This allows designers to easily integrate the two simulations, with SystemC AMS acting as the master of the simulation. At the beginning of the simulation, SystemC AMS instantiates the GVSoC ISS together with SystemC-AMS components. The subsequent simulation proceeds by alternating the execution of GVSoC (through invocations of the `step_until()` API function) and SystemC-AMS. Whenever the SW executing in GVSoC needs to communicate with a bus-connected functional component in SystemC-AMS, the request is intercepted by SystemC AMS from GVSoC AXI operations and then propagated to the functional bus. In the case of GVSoC-dependent extra-functional models, information can be extrapolated from GVSoC through dedicated APIs (e.g., `get_instant_power()` for power demand).

This dual-instance approach allows for a synchronized and detailed simulation of both the functional interactions and power consumption dynamics within the system. This proved to allow effective design space exploration, aware not only of the functionality but also of extra-functional dimensions, such as energy autonomousness and optimization [11].

C. SUNRISE for SoC Design Evaluation

SUNRISE - a **S**calable **Un**ified **R**ESTful **I**nfrastructure for **S**ystem **E**valuation - provides users a unified approach to utilizing virtual prototyping solutions, facilitate access to various simulation technologies and boost cooperation by leveraging decentralized compute resources for deployment of simulation workloads and definition of open APIs. HW architects and algorithm developers can easily evaluate SoCs and IPs by using the *Evaluation API* (EvalAPI) whereas the SoC and IP providers can integrate their simulation models using the *System API* (SysAPI). SUNRISE can be used throughout the development process as different types of simulation models are supported. Figure 5 shows an overview of the SUNRISE framework. In the TRISTAN project, two successful RISC-V IP integrations have been shown together with Codasip and MINRES.

From an industrial perspective, the adoption of the *Evaluation API* (EvalAPI) will lead to the long awaited industry-wide democratization of custom compute. Using well-defined standards, it is now possible to evaluate the performance of software commits on an ever-increasing variety of IP configurations and instantly compare the outputs through a unified data format, as demonstrated by Codasip. This will enable software-driven hardware development and could open the door to AI-assisted co-design of the HW/SW stack.

The different levels of abstraction, which can be covered by the System API, are prototyped during the TRISTAN project by deploying virtual platforms based on DBT-RISE [13] and SCC components, as well as implementation models based on the TGC cores [14] from MINRES. The approach of a standard API allows a content provider to focus on demonstrating the value in a defined environment while the content user focuses on evaluating the results of the workload.

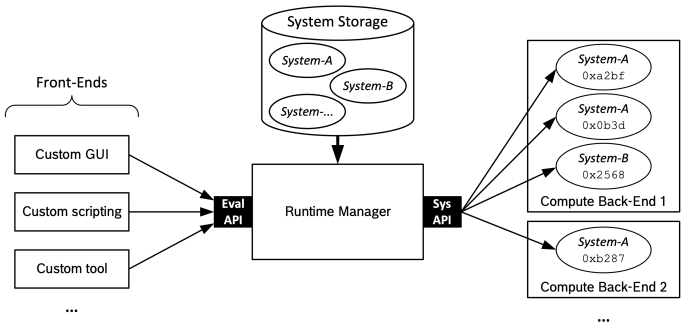

Fig. 5. SUNRISE framework overview. Figure reused from [12].

IV. Tools and Methodologies for Design and Implementation

One key issue in designing and integrating circuits is managing the increasing complexity of today's systems. Methodologies and frameworks to save design time and effort are crucial to reduce manual errors while maintaining flexible but efficient solutions during the design process. To address some aspects of these challenges, this section introduces two approaches developed within TRISTAN focused on efficient IP generation and management. First, we propose a custom instruction acceleration flow based on the commercial tool Catapult HLS. Subsequently, we cover Kactus2, an IP-XACT conform open-source toolset for SoC design and management.

A. High-Level Synthesis HW Acceleration Flow with Catapult

Catapult High-Level Synthesis (HLS) and Verification [15] is a cutting-edge platform for accelerating the IC design flow. Catapult HLS automates the generation of optimized RTL code from C/C++ and SystemC, significantly reducing hardware design time and effort. The tool seamlessly integrates into existing verification flows and enables design space exploration to meet stringent performance, power, and area requirements.

In TRISTAN, we leverage Catapult HLS to develop an application-specific implementation flow for HW acceleration using custom instructions. Our goal is to provide an automated tool to accelerate RISC-V applications while reducing design time and effort for the IC designer. Figure 6 illustrates the flow and high-level architecture of the resulting system. The user provides the flow with a C/C++ algorithm and manually selects the specific custom instruction code sections to be implemented in hardware. Based on this manual custom instruction segmentation, the flow partitions the algorithm into C/C++ hardware accelerator functions and the remaining C/C++ SW code. The accelerator functions are embedded into a C/C++ wrapper and subsequently converted into an RTL description utilizing Catapult. The resulting HDL wrapper accelerator is compliant to OpenHW Group's Core-V eXtension interface specification (CV-X-IF) [16], thus can be integrated with any RISC-V core supporting the interface (as indicated in Figure 6). GCC is used to generate the binary for the selected RISC-V core. Within TRISTAN, we utilize the Core-V cores CVA6 [6] and CV32E40X [17] to enhance and test the generated wrappers.

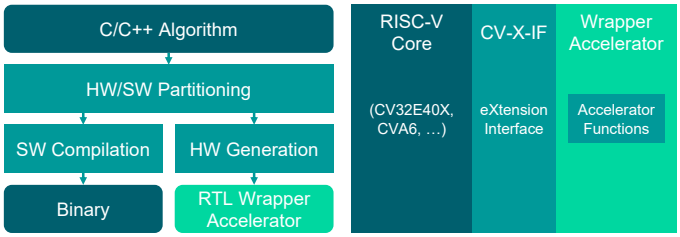

Fig. 6. Flow diagram (left) of the proposed Catapult-based HLS flow, and high-level block diagram (right) of the resulting HDL system consisting of the selected RISC-V core and flow-generated wrapper accelerator.

B. IP Composition with IP-XACT & Kactus2

The reuse of IPs is a more and more prominent part of SoC design through both the fabless design companies as well as the availability of open source IPs. IP-XACT [19] standard aims to formalize IP data exchange format for reuse, integration, and automation. Formalized data allows integration to be done more easily by ensuring that correct information is documented. For IP, IP-XACT information describes the structural model, memory content, and connectivity. This standard also defines a formal generator interface that enables tools to more easily automate the integration of IPs and use flows as well as the generation of design, documentation, and verification for them. IP-XACT defines a standardized way to implement vendor extensions to enable users to add any custom information.

Kactus2 [20] is an IP-XACT based open-source toolset for designing SoCs. Kactus2 enables users to either package their existing IPs with IP-XACT format or to create new ones directly in it. Packaged IPs can be used with graphical hardware design tools to create hierarchical designs that can be generated into HDL. Diagram connectivity for memory-mapped designs enables users to generate HW/SW co-design files, such as memory address headers, device trees, system view description (SVD), and Renode platform files. This enables designers to gain additional benefits from conforming to standards on top of SoC structural modeling and documentation. The scope of IP-XACT and Kactus2 is depicted in Figure 7.

V. Tools and Methodologies for Formal Verification

Two different formal verification workflows of RTL designs, used for different purposes, have been proposed in TRISTAN. The open-source µArchiFI tool is used to evaluate security properties, while the commercial OneSpin tool is used to evaluate the functionality of hardware designs. This section focuses on the µArchiFI tool.

Fault-Injection (FI) consists in applying abnormal physical stress to the HW to modify the behavior of the microelectronics. This leads to the appearance of incorrect values called *faults* in the micro-architecture as detailed by Yuce [21]. These faulty values can be recovered or propagated through the processor circuit and potentially lead to observable effects at the software level. For instance, this can result in skipping an instruction, reading or writing to a wrong address in memory [22]. The

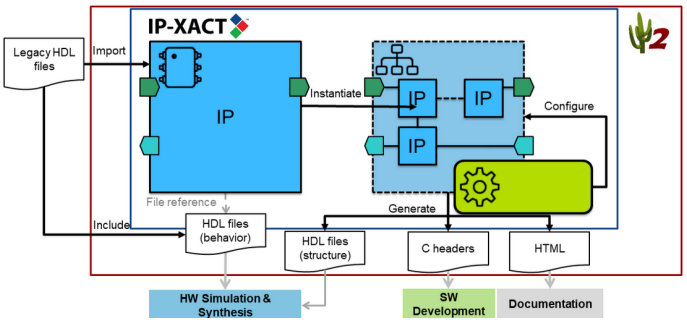

Fig. 7. Scope of IP-XACT defined with blue and Kactus2 with red. Figure reused from [18].

observable effects of the faults can then be exploited by an adversary. To comprehensively locate and characterize FI-based vulnerabilities, developing a formal model of the processor helps to identify which FI attacks, targeting the hardware, defeat a given security requirement often expressed at the software level.

µArchiFI [23] is an open-source tool dedicated to the formal modeling and verification of microarchitecture-level fault injections and their effects on complex HW/SW systems. Such a system-level model requires:

1) The *HW model* of the processor provides a cycle-accurate formal model of both the combinatorial and sequential logics constituting a processor design.
2) The *SW model* describes the sequence of instructions, represented in a binary form and applied as constraints on the initial state of the HW model.
3) The *fault model* indicates how the physical attack interferes with the HW model. A fault model has three dimensions: *temporal*, *spatial*, and *effect*. The *temporal* dimension specifies the targeted clock cycles by the attack. The *spatial* dimension describes which signals can be modified by the fault. Finally, the *effect* dimension defines which values can be applied to the faulty signals.
4) The *security property* is necessary to identify if an FI attack can lead to new vulnerabilities by encoding the expected software behavior.

The Yosys toolchain [24] generates the HW model for different model-checking environments. We demonstrated the practical use of µArchiFI on RISC-V use cases using state-of-the-art model-checking tools for HW verification [23], [25]. We have extended this approach with a notion called *k-fault-resistant partitioning* to solve the fault propagation problem when assessing redundancy-based HW countermeasures [26]. We applied this methodology to the OpenTitan secure element and formally prove the security of its CPU's HW countermeasure to single bit-flip injections. Besides that, we demonstrated that previously intractable problems, such as analyzing the robustness of OpenTitan running a secure boot process, can now be solved by a co-verification methodology that leverages k-fault-resistant partitioning.

VI. Conclusion

This paper provides the current state of the TRISTAN EDA tools landscape, highlighting a selection of proposed tools and methodologies. The goal is to build a comprehensive set of tools in which open-source and commercial solutions co-exist to drive the European RISC-V ecosystem forward. On the one side, open-source tools and frameworks foster customization and interoperability across different flows and enable collaborative research and development. On the other side, established commercial tools with reliable maturity levels remain essential, in particular for the critical stages of the design process.

Acknowledgments

This work was funded by the European Union's Chips JU grant agreement No. 101095947 (TRISTAN - Together for RISc-V Technology and ApplicatioNs), Dec. 2022 - Nov. 2025.

References

[1] European Union, "The European Chips Act," https://eur-lex.europa.eu/.
[2] TRISTAN project CHIPS-JU nr. 101095947, "TRISTAN - Expand, mature and industrialize the European RISC-V ecosystem," https://tristan-project.eu.
[3] Open Source Hardware & Software Working Group, "Recommendations and roadmap for european sovereignty on open source hardware, software and RISC-V," https://ec.europa.eu/newsroom/dae/redirection/document/89438.
[4] N. Bruschi, G. Haugou, G. Tagliavini, F. Conti, L. Benini, and D. Rossi, "GVSoC: A Highly Configurable, Fast and Accurate Full-Platform Simulator for RISC-V based IoT Processors," in *2021 IEEE 39th International Conference on Computer Design (ICCD)*, 2021, pp. 409–416.
[5] C. Andriamisaina, K. Trabelsi, and P.-G. Le Guay, "A methodology for fast and efficient ml-based power modeling," in *2024 IEEE 42nd International Conference on Computer Design (ICCD)*, 2024.
[6] OpenHW Group, "CVA6 RISC-V CPU," https://github.com/openhwgroup/cva6.
[7] F. Bellard, "QEMU, a fast and portable dynamic translator." in *USENIX annual technical conference, FREENIX Track*, vol. 41, 2005, p. 46.
[8] Accelera. (2024) Systemc ams. [Online]. Available: https://systemc.org/overview/systemc-ams/
[9] F. Pecheux *et al.*, "Systemc ams based frameworks for virtual prototyping of heterogeneous systems," in *IEEE ISCAS*, 2018.
[10] S. Vinco, Y. Chen, F. Fummi, E. Macii, and M. Poncino, "A layered methodology for the simulation of extra-functional properties in smart systems," *IEEE TCAD*, 2017.
[11] M. A. Hamdi, G. Pollo, M. Risso, G. Haugou, A. Burrello, E. Macii, M. Poncino, S. Vinco, and D. J. Pagliari, "Integrating SystemC-AMS power modeling with a RISC-V ISS for virtual prototyping of battery-operated embedded devices," in *Proc. of Computing Frontiers (CF))*, https://github.com/eml-eda/messy, 2024, p. 51–54.
[12] T. Kraus, A. Sauer, and I. Feldner, "Deployment of containerized simulations in an API-driven distributed infrastructure," in *DVCon Europe 2024*, 2024.
[13] E. Jentzsch, "MINRES Technologies GmbH; a flexible simuation model for RISCV processors," https://github.com/Minres/DBT-RISE-RISCV.
[14] ——, "MINRES Technologies GmbH; a flexible implementation of RISCV based processors," https://github.com/Minres/CoreDSL.
[15] Siemens EDA, "Catapult High-Level Synthesis and Verification," https://eda.sw.siemens.com/en-US/ic/catapult-high-level-synthesis/.
[16] OpenHW Group, "Core-V eXtension interface (CV-X-IF)," https://github.com/openhwgroup/core-v-xif.
[17] OpenHW Group, "Core-V CV32E40X RISC-V IP," https://github.com/openhwgroup/cv32e40x.
[18] Tampere University System-on-Chip Research Group, "Introduction to ip-xact and kactus2 - kactus2/kactus2dev wiki," https://github.com/kactus2/kactus2dev/wiki/.
[19] "Ieee standard for ip-xact, standard structure for packaging, integrating, and reusing ip within tool flows," *IEEE Std 1685-2022 (Revision of IEEE Std 1685-2014)*, pp. 1–750, 2023.
[20] A. Kamppi, L. Matilainen, J.-M. Maatta, E. Salminen, T. D. Hamalainen, and M. Hannikainen, "Kactus2: Environment for embedded product development using ip-xact and mcapi," in *2011 14th Euromicro Conference on Digital System Design*, 2011, pp. 262–265.
[21] B. Yuce, P. Schaumont, and M. Witteman, "Fault Attacks on Secure Embedded Software: Threats, Design, and Evaluation," *Journal of Hardware and Systems Security*, vol. 2, no. 2, pp. 111–130, 2018.
[22] J. I. Library, "Application of Attack Potential to Smartcards and Similar Devices," Tech. Rep., 2013.
[23] S. Tollec, M. Asavoae, D. Couroussé, K. Heydemann, and M. Jan, "μArchiFI: Formal Modeling and Verification Strategies for Microarchitectural Fault Injections," in *Formal Methods in Computer-Aided Design, FMCAD 2023, Ames, IA, USA, October, 2023*, pp. 101–109.
[24] C. Wolf, "Yosys Open SYnthesis Suite," https://yosyshq.net/yosys, accessed: July 24, 2023.
[25] S. Tollec, M. Asavoae, D. Couroussé, K. Heydemann, and M. Jan, "Exploration of Fault Effects on Formal RISC-V Microarchitecture Models," in *Workshop on Fault Detection and Tolerance in Cryptography, FDTC 2022, Virtual Event / Italy, September 16, 2022*, 2022, pp. 73–83.
[26] S. Tollec, V. Hadzic, P. Nasahl, M. Asavoae, R. Bloem, D. Couroussé, K. Heydemann, M. Jan, and S. Mangard, "Fault-Resistant Partitioning of Secure CPUs for System Co-Verification against Faults," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2024, no. 4, pp. 179–204, 2024.