A Survey of the RISC-V Architecture Software Support

Benjamin William Mezger, Douglas Almeida dos Santos, Luigi Dilillo, Cesar A Zeferino, Douglas Rossi de Melo

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

A survey of the RISC-V architecture software support

BENJAMIN W. MEZGER[1], DOUGLAS A. SANTOS[12], (Student, IEEE), LUIGI DILILLO[2], CESAR A. ZEFERINO[1], and DOUGLAS R. MELO[1], (Member, IEEE)

[1]University of Vale do Itajai, Laboratory of Embedded and Distributed Systems, Brazil
[2]University of Montpellier, Laboratoire d'Informatique, de Robotique et de Microelectronique de Montpellier, France

Corresponding authors: Benjamin W. Mezger (ben@edu.univali.br), Douglas R. Melo (drm@univali.br).

This work was supported in part by the Foundation for Support of Research and Innovation, Santa Catarina (FAPESC) under Term 2021TR001907.

ABSTRACT RISC-V is a novel open instruction set architecture that supports multiple platforms while maintaining simplicity and reliability. Despite its novelty, the software support for RISC-V has been increasing in the last years, given that popular tool-chains and operating systems already have support for RISC-V. However, although many works have been exploring the RISC-V software ecosystem, no work that raised the current state of software support for RISC-V is available. In this context, this survey reviews the contributions introduced in the last years to understand the RISC-V's software ecosystem and its usage in both academic and industrial environments. We classified and evaluated the works into four main categories: fields of application, RISC-V implementations, software architecture, and deployment features. The primary goal of this research is to provide the community with a comprehensive overview of the current state of the art of RISC-V software support and identify and highlight the main contributions of recent works.

INDEX TERMS RISC-V, Software Support, Operating Systems

I. INTRODUCTION

WITH the increasing number of instructions of popular Instruction Set Architectures (ISAs) and the requirements of backward compatibility of older extensions, researchers from the University of California at Berkeley developed an open ISA based on Reduced Instruction Set Computer (RISC) principles. This architecture was named RISC-V [1] and seeks to provide a base ISA and optional application-specific extensions to support software engineers with a small and robust ISA [2]. In other words, it offers a stable ISA for compiler and operating system designers, enabling them to work with hardware designers to provide additional resources to meet application requirements and participate in the decisions and implementations of the RISC-V ISA specifications [2]. Nowadays, RISC-V is maintained by RISC-V International, a non-profit organization.

RISC-V also aims at becoming a universal ISA by supporting numerous processor sizes, from embedded controllers to high-performance computers and a wide variety of software stacks and programming languages [2]. Moreover, given the future computing landscape, multiple platforms such as Internet of Things (IoT) devices and personal mobiles will likely dominate the market, requiring ISAs to support these systems [3].

Over the last decade, several studies have focused on the different issues related to the RISC-V architecture, and part of these contributions is summarized and analyzed in survey papers focused on security [4]–[7] and open-sourcing [8], [9] aspects. Although many studies have addressed the software stack for RISC-V, there is no report on the literature analyzing these studies from a unified perspective. Driven by the growth of the RISC-V's ecosystem and the lack of studies concerning the current state of software for this ISA, this work seeks to fill this gap and explore the state-of-the-art software support for RISC-V architectures. This survey analyzes works carried out in the last five years to understand the RISC-V's software ecosystem and its usage in academic and industrial environments. The main contributions of this research are: (i) providing the community with a comprehensive overview of the current state of the art of the RISC-V software ecosystem, and (ii) identifying and highlighting the main contributions of recent works.

The remainder of this work is organized as follows. Section II summarizes the RISC-V ISA and Section III describes

the methods applied to conduct this review. Next, Sections IV and V cover the field of applications and the RISC-V implementations, respectively. Finally, Sections VI and VII cover a set of software support characteristics. Concluding, Section VIII presents our final remarks.

II. RISC-V

Unlike prior ISAs, RISC-V relies on a modular design by providing a frozen base ISA core (RV32I), which will never change, and extensions that provide further functionalities. Moreover, it supports a full software stack, providing a stable target to compiler writers, operating system developers, and assembly language programmers [2]. A modular design enables optional standard extensions that the hardware can include if specified by the system's requirements, facilitating the development of small or large-scale applications and compilers to generate more reliable code. Besides, when RISC-V needs to include new instructions to the ISA, they are kept optional and non-required for all future RISC-V implementations, contrary to incremental ISAs [2].

A. INSTRUCTION FORMAT

The RV32I base ISA has four instruction formats (R/S/I/U) and two variants (B/J) based on the handling of immediate operands. All of them have a fixed length of 32 bits and must be aligned on a four-byte boundary in memory [10]. Thus, the six instruction formats are:

- R-type, for register-register instructions.
- I-type, for register-immediate and load instructions.
- S-type, for store instructions.
- B-type, for conditional branch instructions.
- U-type, for instructions with a large upper immediate.
- J-type, for unconditional jump instructions.

Given that the six instruction formats have regular encoding, the decoding of instructions is much more straightforward than ARM or x86 architectures. For example, RISC-V provides three register operands at the same position in all formats, simplifying the decoding process. In addition, the specified registers to be read or written are always in the same position in all instructions, enabling register access to start before the instruction decoding phase [2], [11].

B. THE BASE ISA

The RISC-V ISA provides two primary base integer variants: RV32I (for 32-bit) and RV64I (for 64-bit). The base instruction-sets provide the minimum requirements for running a processor and are able to run a simple operating system. It uses a two's-complement representation for signed integer values, and data is stored in memory using the little-endian system, although it allows non-standard alternatives to provide a big-endian memory system [11].

C. PRIVILEGE LEVEL

The hardware must provide the OS mechanisms that enable the processor to change its execution privilege status, such as going from the user mode to the supervisor mode, and provide protection across different software components [12]. RISC-V supports three privilege levels of execution to protect different software stacks, enabling multiple software stacks to run with different privileges levels. Attempts to perform operations not permitted by the current privilege level will result in exceptions and require handling by an underlying execution environment. The highest and mandatory privilege level of the RISC-V hardware platform is the M-Mode, which is inherently trusted and has all low-level access to the system. Besides M-Mode, RISC-V supports Supervisor-mode (S-Mode) intended for OS usage and User-mode (U-Mode) for conventional applications. In addition, each privilege level has a set of privileged ISA extensions with support to optional extension, enabling, for example, having S-Mode extended to support a hypervisor execution environment [11].

D. REGISTERS

RISC-V has 32 registers (`x0-x31`), and the RISC-V's Application Binary Interface (ABI) determines their name. Register `zero` is hardwired to zero and always holds the value zero, mainly to simplify the ISA. The `ra` and `sp` registers hold the return address and stack pointer, respectively. Registers `t0-t6` holds temporary values that are not guaranteed to persist after a function call, and `s0-s11` hold persistent values across function calls. Finally, registers `a0-a1` hold the first two arguments of a function and return value, and `a2-a7` hold any remaining arguments [2].

E. CONTROL AND STATUS REGISTER

The Control and Status Registers (CSRs) are system registers to control and monitor the machine's current state. CSRs can be read or written through specific CSR instructions, and have restrictions for low-privilege levels. A RISC-V implementation may contain additional CSRs, accessible to a subset of the privilege levels. Any attempt to either access a CSR without the appropriate privilege level or to write into a read-only CSR raises an illegal instruction exception [13].

F. EXCEPTIONS AND INTERRUPTS

Both exceptions and interrupts on RISC-V processors are considered traps. As exceptions and interrupts happen during runtime, the processor provides mechanisms to make an unscheduled procedure call to an arbitrary address [12]. RISC-V classifies traps into two main categories: synchronous and asynchronous. Synchronous traps are exceptions and result from an instruction execution, such as accessing an invalid memory address or executing an invalid instruction. Asynchronous traps are interrupts and are external events that occur asynchronously to the instruction stream. RISC-V sets the most significant bit of the `mcause` control status registers to identify whether the trap is synchronous or asynchronous, and the least significant bits for identifying which interrupt or exception occurred [2].

RISC-V has three sources of interrupts: software, timer, and external interrupts. Software interrupts enable programmers to interrupt a given CPU, allowing efficient Inter-Process Communication (IPC), while timer interrupt is triggered when a hart (i.e., hardware thread) time comparator exceeds or equals the global `timebase` register. External interrupts, in turn, are asserted by a platform-level interrupt controller [13]. The mechanisms for raising and clearing interrupts can vary according to the hardware platform, given they can use different memory maps and demand divergent features from their interrupt controller [2]. However, all RISC-V systems handle exceptions and mask interrupts in the same way.

G. ADOPTION AND MOTIVATION

RISC-V Foundation is a nonprofit organization and counts with more than 40 sponsoring companies [12]. For the past decade, NVIDIA has been shipping their Graphics Processing Units (GPUs) with proprietary microcontrollers called Falcon, but since 2016, NVIDIA is evaluating RISC-V for their GPUs [14]. Western Digital open-sourced the SweRV Core, an industry-qualified processor featuring a 32-bit in-order, 2-way superscalar design with a 9-stage pipeline core [15]. In 2019, Alibaba revealed its first embedded RV64GCV RISC-V-based processor, 64-bit, high performance, and with a set of custom extensions [16], [17]. Due to the recent US restrictions on its ARM design, Huawei HiSilicon released their first RISC-V-based board together with the Harmony OS in May 2021 [18]. With LEON SPARCv8 being the dominant architecture in European avionics and facing difficulties in leveraging software from the commercial domain, the Dependable Real-time Infrastructure for Safety-critical Computer (De-RISC) project introduces the RISC-V architecture for aviation and space environments by using with fault-tolerant techniques and supporting compute-intensive applications [19]. Further, apart from Apple's recent switch to ARM-based System On Chip (SoC), it has recently demonstrated an interest in exploring the RISC-V architecture [20].

Apart from industrial motivation and adoption, academic development is also in progress. For example, the PULPino processor developed at the Swiss Federal Institute of Technology (ETH Zürich) is ready for industrial standards. PULPino is optimized for low power consumption, concentrating on providing IoT solutions [21], [22]. Given this background, the institution of technology Semico Research Corp. estimates that in 2025 the market will have around 62.4 billion RISC-V cores worldwide [23], [24], as represented by the graph in Figure 1.

III. MATERIALS AND METHODS

In order to carry out this survey and identify the current state of the art of RISC-V software support, we conducted a bibliographic survey using the following materials and methods. First, we performed a search on the IEEE, ACM, Springer, and Usenix digital libraries using this search query: `('RISC-V' AND (Software OR 'Operating System' OR OS))`, and limited the search to retrieve works published in the last five years. Furthermore, we applied the impact of the publication channel and the normalized number of citations received by each work to select the ones to be analyzed.

After the search and selection phase, we identified the following RISC-V software ecosystem shown in Figure 2. From top to bottom, we first explore the variety of environments RISC-V is applied to and then continue to explore available implementations that have been proposed. We then further explore the different software architectures that have been ported or implemented on RISC-V and the deployment characteristics, such as security, reliability, and power features.

This work will further explore these categories in the following sections: field of application (Section IV); RISC-V implementations (Section V); and software support, including software architecture, OS support, file systems, network stack, and uncategorized features (Section VI). In addition, we analyzed the additional features, including the support for security, reliability, and low-power operation (Section VII).

IV. APPLICATION FIELDS

The space industry has had difficulties leveraging software from the commercial domain and is now considering alternative architectures in a larger commercial market [19]. Furthermore, with the introduction of SoC and multi-core processors, this industry now seeks to migrate components to a higher level of integration, introducing a new set of issues that needs to be solved. In this sense, a modular architecture, such as RISC-V, enables designers to extend the processor and implement functionalities tailored to their application requirements, making RISC-V highly adaptable to various environments.

The De-RISC project introduces a novel RISC-V hardware/software platform meeting the requirements and functionalities imposed by the space environment. De-RISC meets reliability by designing its Multiprocessor System-on-Chip (MPSoC) with fault-tolerance techniques to provide a correct operation in the presence of faults [25]. The SELENE project, depicted in Figure 3, proposes a reliable platform for safety-critical computing. It is built on open-source technology and enables designers to adapt the system to a specific requirement domain, integrating applications of different criticalities and demands, and achieving a diverse set of redundancy and performance [26]. The openness of RISC-V improves component reuse and prevents the need for developing projects from scratch, increasing productivity and reducing cost [27].

With the increase of embedded device deployments and the expected growth of the Industrial Internet Of Things (IIoT) market to reach 1.11 trillion US dollars by 2028 [28], [29], Internet of Things (IoT) devices have become a fundamental part of the lives of billions of people around the world. Securing these devices without loss of performance or increasing power usage has become a matter of discussion with the accelerating growth of connected devices. A set

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3174125, IEEE Access

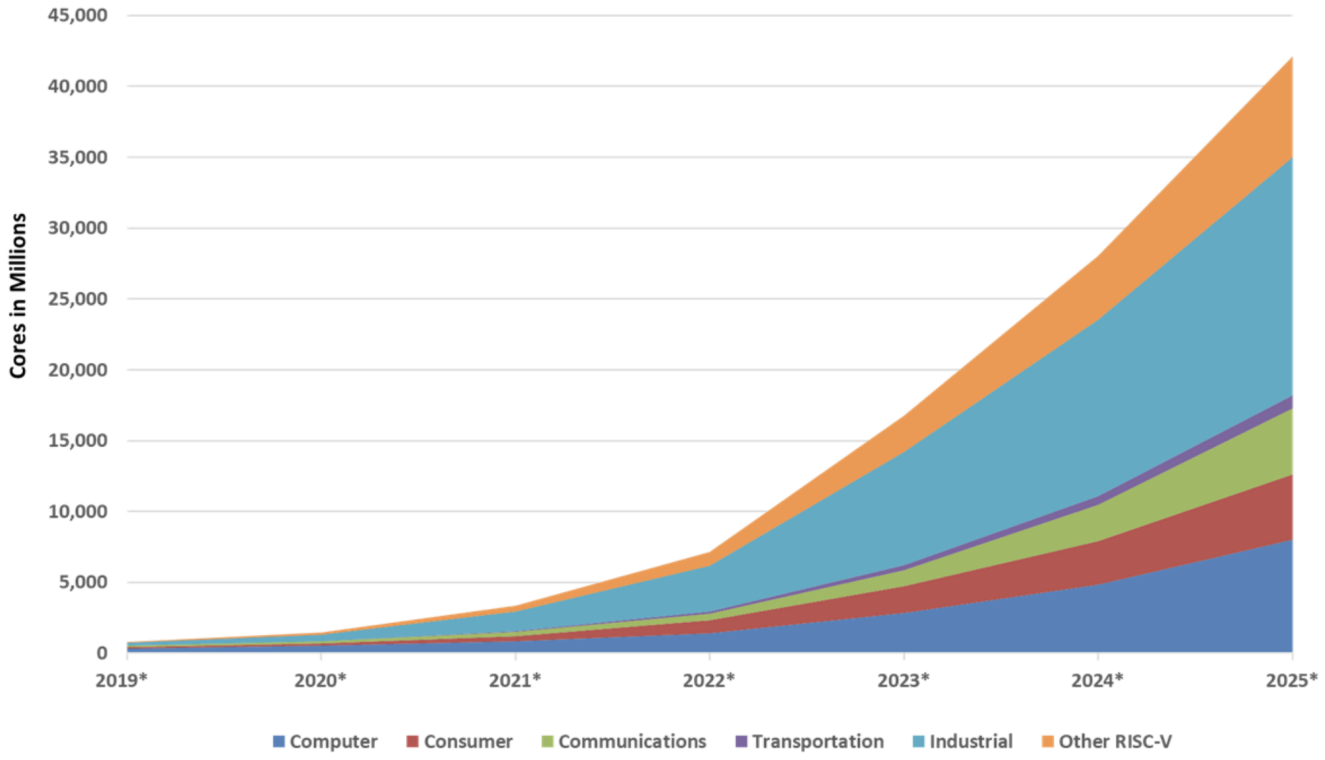

FIGURE 1. Market forecasting for RISC-V [24].

of security research for constrained devices is embraced by the RISC-V community and investigated by the scientific community. Solutions addressing high-level goals identified by NIST IR 8228 [30], authenticated secure-boot images while verifying updates during runtime [31], hardware enforcements [32], memory isolation mechanisms, and post-quantum solutions [33] are all available for the RISC-V architecture.

Apart from security and reliable applications, RISC-V has a set of research fields concerning AI-based applications. In [34], the authors seek to provide a heterogeneous processor design for Convolutional Neural Network (CNN)-based applications by proposing a domain-specific architecture design and an accelerator for the inference of CNNs, adhering to low-power characteristics. The SiFive Intelligence X280 is a multi-core RISC-V compliant processor that optimizes Artificial intelligence and Machine Learning (AI/ML) inferencing computing. X280 uses the vector extension and SiFive Intelligence extensions, making the core suitable for high-throughput single-threaded and power-constrained applications [35]. Further, the work of [36] aims at providing system call isolation characteristics for embedded devices.

RISC-V is highly adaptable for various contexts, given the open specification, its modularity, and the community. It enables designers to deploy from small, low-cost embedded devices to a large-scale system with custom extension support meeting the requirements imposed by the system.

V. RISC-V IMPLEMENTATIONS

Despite the requirements of a general-purpose processor, RISC-V enables domain-specific implementations to exist, either by implementing the RISC-V base ISA with custom characteristics or implementing a new extension.

A. PROCESSOR SOFT-CORE

The PULPino project provides three RISC-V core implementations: (i) a 32-bit 2-stage pipeline named Ibex (formally named Zero-riscy), (ii) a 4-stage pipeline core named CV32E40P (formally named RI5CY), and (iii) a 64-bit 6-stage core named CVA6 (formerly named Ariane). Ibex, maintained by lowRISC and illustrated in Figure 4, is well suited for embedded control applications and supports the Integer (I) or Embedded (E), Integer Multiplication and Division (M), Compressed (C), and Bit Manipulation (B) extensions, and is available as a SystemVerilog project [37], [38]. The CV32E40P core, depicted in Figure 5, implements the RV32IMFC ISA and the Xpulp extension for higher code density, performance, and energy efficiency characteristics [37], [39]. Finally, the CVA6 core, illustrated in Figure 6, focuses on reducing the critical path, implements the RV64IMAC ISA, and supports a configurable size and separate Translation Lookaside Buffer (TLB) [40].

The work of [41] injects faults on a 64-bit, 5-stage pipeline, LowRISC v0.2 Rocket core processor RTL to leak and highlight the importance of hidden registers in the processor pipeline. The authors of [42] use the RISC-V Rocket core

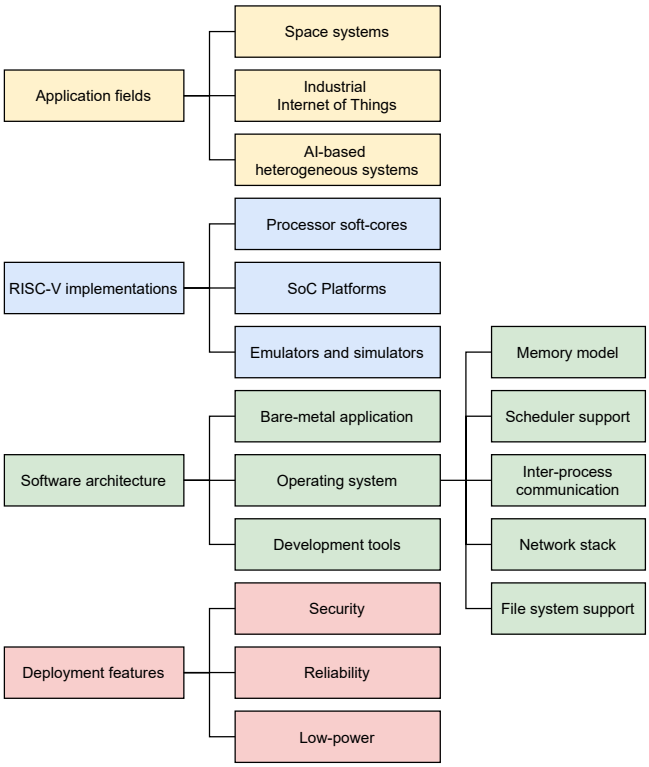

FIGURE 2. RISC-V ecosystem overview.

version 1.7 to provide a kernel hardware-based monitoring platform and a hardware interface architecture to overcome the semantic gap problem. Šišejković et al. [43] implement a hardware/software solution using the 64-bit 6-stage in-order, single-issue Ariane Core. In their work, [44] adopts a CV32E40P core from PULP platform to propose initial ideas on using control logic drivers to create subsystems and system-level portable models from existing block-levels. The work seeks to help verification engineers with vertical reuse by defining a set of algorithms that will semi-automate transformations of portable models for different scenarios.

SonicBOOM, illustrated in Figure 7, is a Register-Transfer Level (RTL) implementation of a RISC-V RV64GC superscalar out-of-order core written in Chisel3. SonicBOOM improves the architectural characteristics of BOOMv2 and seeks to provide a state-of-the-art platform for high-performance research. The SonicBOOM core optimizes the execution path and redesigns the instruction fetch unit with a hardware Tagged Geometric (TAGE) branch predictor algorithm. Further, SonicBOOM's load-store unit provides multiple loads per clock cycle and achieves 6.2 CoreMark/MHz [45], [46].

Furthermore, [47] uses the RISC-V BOOM implementation to provide a platform for Robot Operating System (ROS)-based robotic applications.

In June 2021, SiFive announced the P270 and the P550 cores to their Performance family. P270 is an 8-stage, dual-issue, efficient in-order pipeline compatible with RISC-V's RV64GCV ISA. P550 features 13-stage, triple-issue, out-of-order pipeline compatible with RISC-V's RV64GC ISA and delivers a SPECInt 2006 score of 8.65/GHz, delivering the highest performance RISC-V processor available today [48]–[50].

NOEL-V is a synthesizable VHSIC Hardware Description Language (VHDL) processor with support to the 32- and 64-bit RV{IM,IMAC,GCH} ISA, with single- or dual-issue features. The core is available as part of a subsystem that includes system peripherals and is configurable to use the supports RISC-V extensions upon configuration. The NOEL-V dual-issue processor allows two instructions per clock cycle to execute and implements advanced branch prediction capabilities [19], [51].

In [34], the authors implement their RISC-V CPU based on the low-power 2-stage pipeline Hummingbird E200 RISC-V core, which implements the RV32IMAC ISA. Further, in order to explore and present a compatible RISC-V with Trusted Execution Environment (TEE) featuring security algorithm accelerators, [52] uses a 64-bit RISC-V with the IMAFDC ISA implementation, along with a Rocket chip generator as a hardware platform to integrate the security accelerator.

B. SOC PLATFORMS

The PULPino project provides a set of complete system platforms: (i) PULPissimo and PULPino, two platforms based on a single-core microcontroller, (ii) OpenPULP, a multicore IoT processor, and (iii) Hero, a multi-cluster heterogenous accelerator, which combines PULP-based parallel manycore accelerator on a FPGA with a hard ARM Cortex-A multicore host processor, all four illustrated in Figures 8 to 11, respectively.

To encourage SoC security research, [56] employs the Ariane SoC RTL to investigate and discover hardware vulnerabilities in SoC designs. In order to further explore microarchitectural security, [57] conducts two case studies on the Ariane and PULPissimo SoC. Moreover, [58] evaluates the impact of OS on the reliability of a RISC-V-based SoC by using a LowRISC implementation on a Xilinx FPGA.

The work by [59] uses a LowRISC FPGA implementation to propose a hardware/software-based solution to secure system integrity. Wang et al. [60] present an open-source framework for easy deployments of Neural Network (NN)s trained with the Fast Artificial Neural Network (FANN) library on an ARM Cortex-M core as well as on a RISC-V-based PULP processor.

The FE310-G000 SoC by SiFive, illustrated in Figure 12, has an E31 core with a single-issue in-order pipeline and implements the RV32IMAC ISA. The E31 core peaks a sustainable execution rate of one instruction per clock cycle. FE-3100-G000 features two Universal Asynchronous Receiver-Transmitter (UART) devices for serial communication, three Quad Serial Peripheral Interfaces (QSPIs), three Pulse-Width Modulation (PWM) ports, and support to low-power operations and wakeup [61]. The work of [62], which proposes a set of novel approaches that enhance Virtual Prototype (VP)

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3174125, IEEE Access

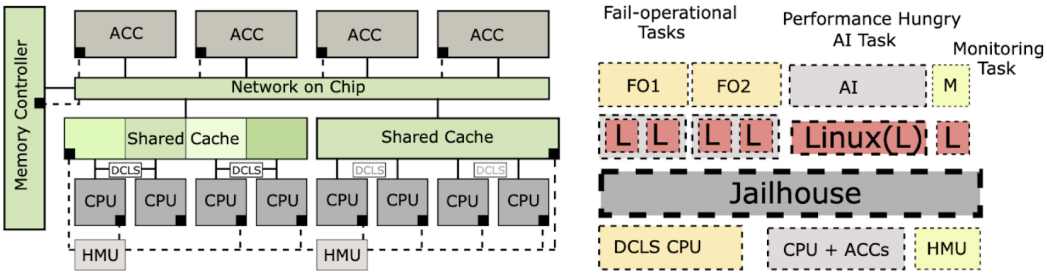

FIGURE 3. The architecture of the SELENE project [26].

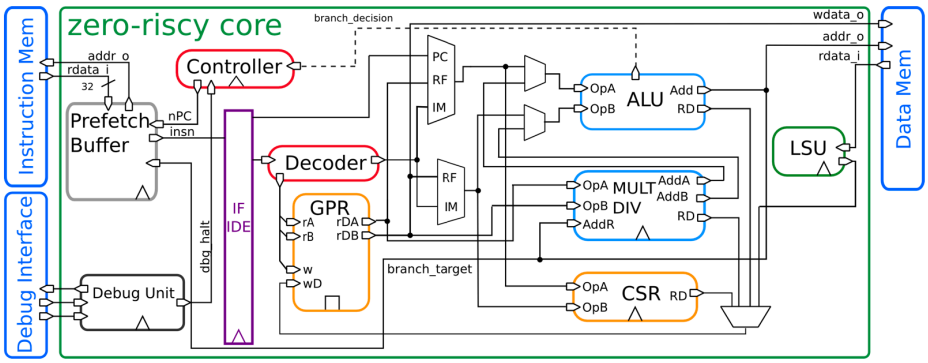

FIGURE 4. Block diagram of the Ibex core [37].

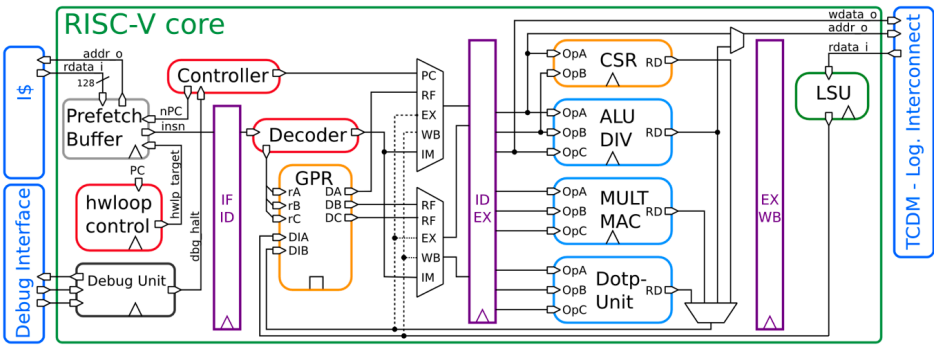

FIGURE 5. Block diagram of the CV32E40P core [37].

design flow, supports the HiFive1 board, which features the FE310-G000 SoC. Similarly, [63] proposes an automated quality-driven methodology which supports the HiFive 1 board.

Du et al. [64] use the SiFive U500 SoC with the RV64IMAFD ISA to add support for a secure and efficient cross-process call architecture.

The PolarFire SoC FPGA by Microchip is a low-power, thermal efficient, and defense-grade security for intelligent, connected systems. The SoC has a 5-stage single-issue in-order pipeline RISC-V and does not suffer from Meltdown and Spectre exploits of common architectures. PolarFire cores are deterministic and coherent with the memory sub-system, enabling Linux and real-time capable applications to execute. The SoC implements the RV64GC and RV64IMAC ISA [65].

The work of [31] uses a Universal Sensor Platform (USeP) SoC to meet the requirements imposed by IoT, which features a RISC-V processor and integrates a range of peripherals with a scalable subsystem as a Three Dimensional System-in-Package (3D-SiP). Further, the Sipeed MAIX-Go development board employed by [66] features a Kendryte SoC K210, with a dual-core 64-bit RISC-V with the RV64GC ISA, including an Floating-Point Unit (FPU) compliant with support to single and double-precision multiply, divide, and square-root operations.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3174125, IEEE Access

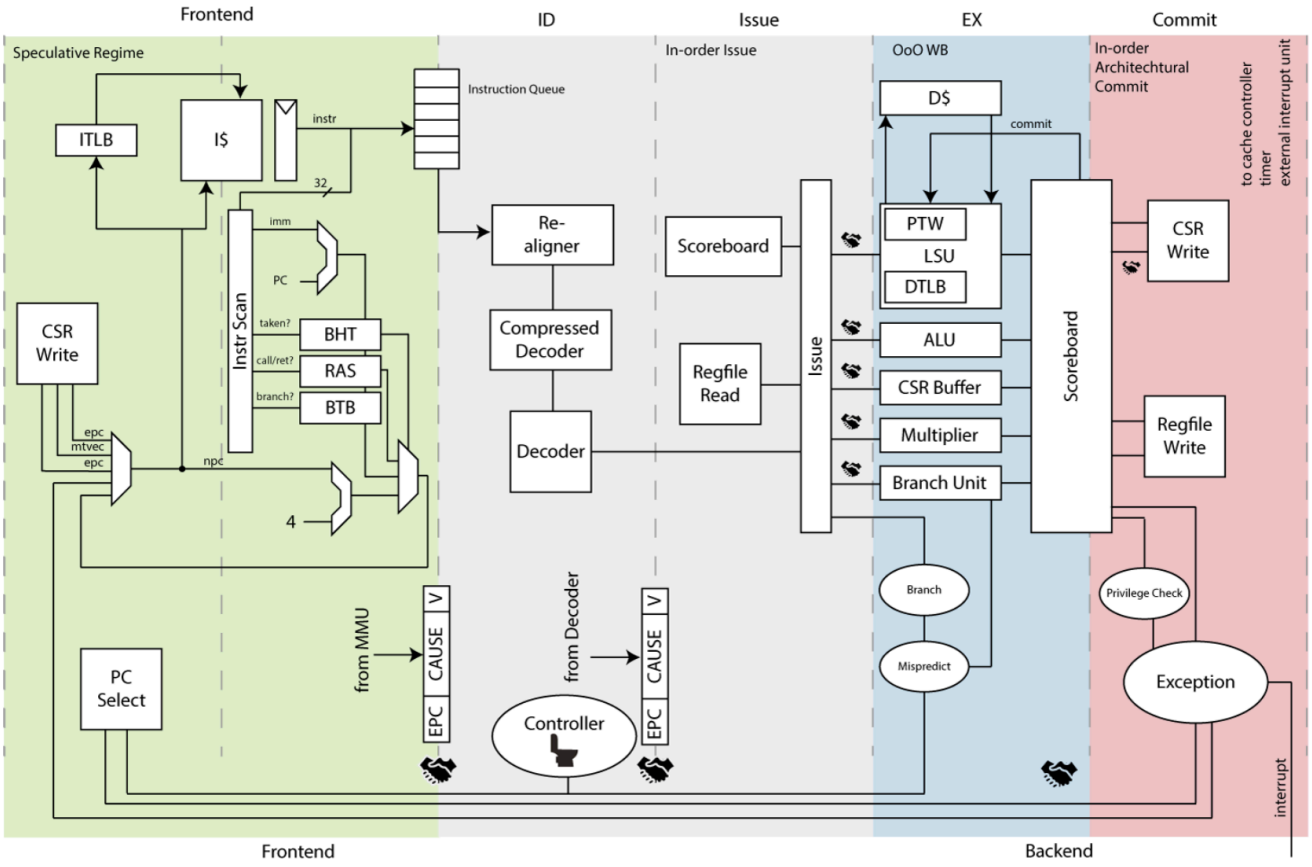

FIGURE 6. Block diagram of the Ariane core [40].

C. EMULATORS AND SIMULATORS

The open-source QEMU project is a generic machine emulator and virtualizer, providing a virtual model of an entire machine to run a guest OS. QEMU supports 32- and 64-bit RISC-V implementation and several different machines, including Microchip's PolarFire SoC, SiFive's HiFive Unleashed, and Shakti C platform [67], [68].

The Gem5 is an open-source modular simulator for computer architecture research, including system-level architecture and processor microarchitecture. Gem5 support to RISC-V privileged ISA specification is still in development [69].

Spike is a RISC-V ISA simulator and implements a functional model of one or more RISC-V hardware threads. It supports a large set of RISC-V extensions and eases simulating new instructions by letting the user describe the functional behavior and add the opcode and opcode masks [70].

Furthermore, the RISC-V Assembler and Runtime Simulator (RARS) assembles and simulates the execution of the RISC-V assembly language. RARS is built on top of the MIPS Assembler and Runtime Simulator (MARS) and extends the software to enable features such as instructions hotload of RISC-V extensions [71]. Similarly, [72] provides a web-based server-side simulation of a 5-stage pipeline RISC-V implementation for use in classrooms. The application enables writing simple assembly programs and visualizing data in registers, memory, and the internal state of the pipeline.

D. DISCUSSION

This section reviewed different RISC-V implementations and applications to comprehend the RISC-V ecosystem and the available research towards the open ISA. Given RISC-V's open specification, multiple implementations exist and tackle specific problems for a given field of application, making RISC-V highly adaptable. Besides the presented RISC-V cores and SoCs, there are several others not mentioned in this work that have shown to be production-ready, supporting low to high-performance computing requirements.

VI. SOFTWARE SUPPORT

The ISA interface encompasses the machine language instructions that a computer can run, acting as a boundary between the hardware and software layer [73], represented in Figure 13. In RISC-V's privilege layer, both the user application and the OS can access the ISA directly, as RISC-V provides a subset of instruction repertoire per layer.

The Application Binary Interface (ABI) defines a standard for binary portability across programs by defining the system

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3174125, IEEE Access

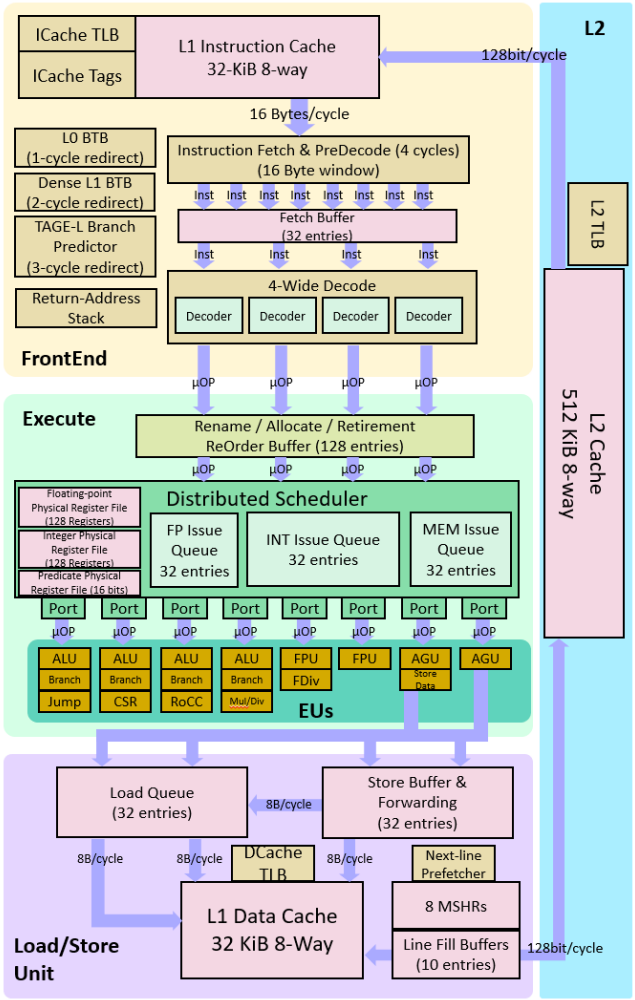

FIGURE 7. Block diagram of the SonicBOOM core [45].

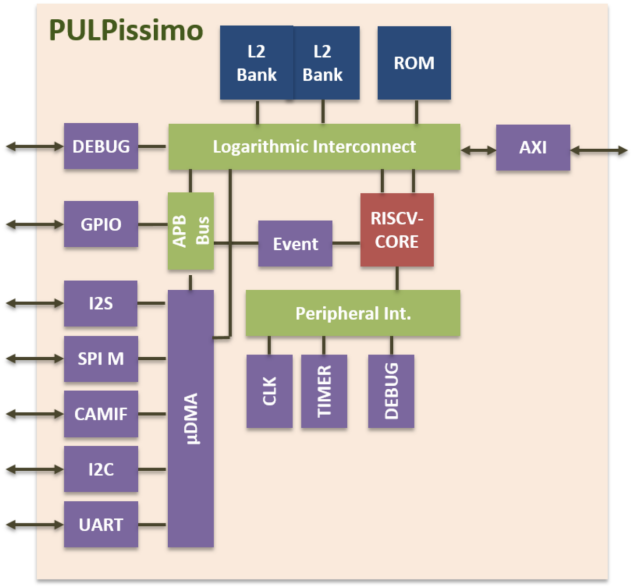

FIGURE 8. Block diagram of the a PULPissimo SoC [53].

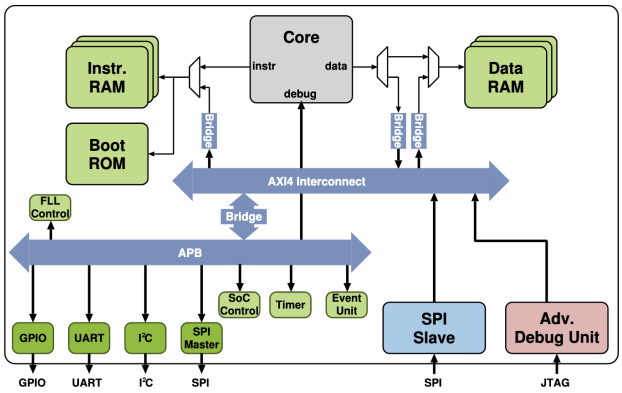

FIGURE 9. Block diagram of the a PULPino SoC [37].

call interfaces to the OS and hardware resources available in the system through the ISA [73]. This allows binaries compiled to a specific ABI to run without modifications in a different system with the same ISA and OS.

RISC-V compiler ABI depends on the RISC-V extensions present; the `ilp32`, `ilp32f`, and `ilp32d` are all RV32 ABIs which depend on the RISC-V extension. The `ilp32` ABI refers to a 32-bit size `int`, `long`, and pointer data type in C-language, while the optional suffix refers to how floating-point arguments get stored in a register. The `ilp32` ABI stores all floating pointers in the integer-type register file, while `ilp32f` stores simple floating-point arguments in the floating-point-type register file. The `ilp32d` ABI applies to the 32-bit data type with double floating-point precision stored in the floating-point-type register file [2].

The following subsections seek to evaluate and review works that address different types of software architecture to understand the scope of end-user application and Operating System (OS) support.

A. SOFTWARE ARCHITECTURE

The architecture of software relates to its structure and how these components are separated, and their interrelationships. Software engineers seek to structure software to meet current and future demands, making the system reliable, manageable, adaptable, cost-effective, and scalable [74].

1) Bare-metal application

Applications running on a bare-metal systems have direct access to the processor and peripherals. These applications are not managed by an OS layer, enabling applications that require runtime guarantees along with constraint hardware to function as expected.

The work of [75] extends their previous work on COmpiler Assisted Software Fault Tolerance (COAST) [76], which seeks to explore Commercial off-the-shelf (COTS) systems to provide an automated compiler modification, bringing

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3174125, IEEE Access

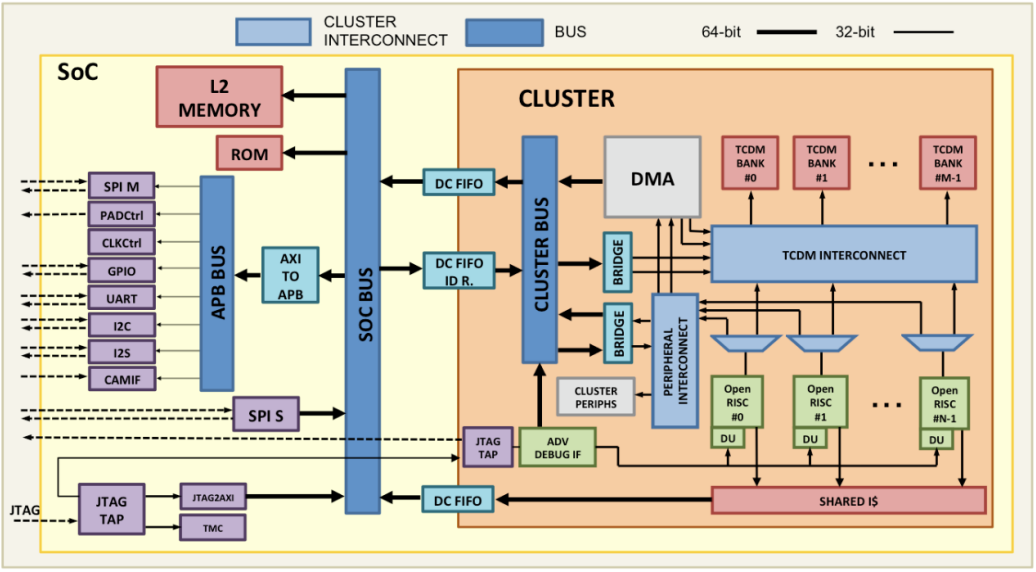

FIGURE 10. Block diagram of the a OpenPULP SoC [54].

support to several new processing platforms, such as RISC-V and Xilinx SoC-based products. The compiler inserts dual- or triple-modular redundancy during the compilation phase, enabling the software to correct errors during runtime. The proposal is attractive for applications requiring tolerance against Single Event Effect (SEE) and is well-suited for processing in a high radiation environment. Although the tool provides both Duplicate With Compare (DWC) and Triple Modular Redundancy (TMR) mechanisms, the default configuration uses Variables 3 (VAR3) protection mechanism. Figure 14 illustrates the COAST architecture.

Stahl et al [77] propose a driver development flow to ease the definition of hardware/software interfaces without a fixed register layout and reduce the development effort and memory usage of an Microcontroller Unit (MCU). The proposal uses a Domain-Specific Language (DSL) to describe the behavior using features such as bit field arrays and hierarchy and provides a custom C-like struct definition for grouping the bit fields of a driver. In addition, the work proposes a heuristic, code analysis, and generation technique to find an optimized register layout that exploits its performance and memory footprint. The proposal uses PULPino General-purpose Input/Output (GPIO) and Serial Peripheral Interface (SPI) drivers and has reduced the estimated run time by 52%, 32% reduction of memory accesses, while the driver code size is reduced by 22%.

Fell et al. [78] explores time side-channel attacks and investigates the impact of source-code obfuscation techniques to verify what information leakage is exploitable. The work proposes two techniques to mitigate timing leakage in obfuscated codes: a compiler-based technique named TAD (Time side-channel Attack Defense), which consists of providing an extension to the LLVM compiler, removing conditional branches, and replacing primitive instructions with custom instructions that manifest non-deterministic executing time during runtime.

2) Operating systems

Zhang et al. [79] propose an automatic kernel code synthesis and verification technique framework, which seeks to build a verifiable OS kernel with a high degree of proof automation and a low burden of proof. Moreover, the proposed framework enables a software developer to write the required specification of the kernel and translates the corresponding specification to C code. The authors have provided a kernel named iv6, which combines exokernel architectural aspects with characteristics of seL4, a mathematically proven and trustworthy microkernel Šišejković et al. [43]. The kernel initializes in RISC-V's M-Mode, runs in a separate address space from userspace, and uses identity mapping techniques for the kernel along with additional characteristics.

In addition to the OS architecture, [80] explores the building blocks of the development cycle of an open-source OS for the 64-bit little-endian RISC-V architecture, proposing a custom Linux distribution from Linux From Scratch (LFS) with independent userspace and a package manager written in Lua programming language. The proposal seeks to provide a Linux-based distribution for open source hardware and is the first OS targetting RISC-V.

Implementations of a RISC-V architecture with hardware/software co-design are also available, such as the work proposed by [81], which captures possible application-kernel interaction as an Finite-state Machine (FSM) and integrates the Real-time Operating System (RTOS) semantics directly into the processor pipeline. The proposal significantly im-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3174125, IEEE Access

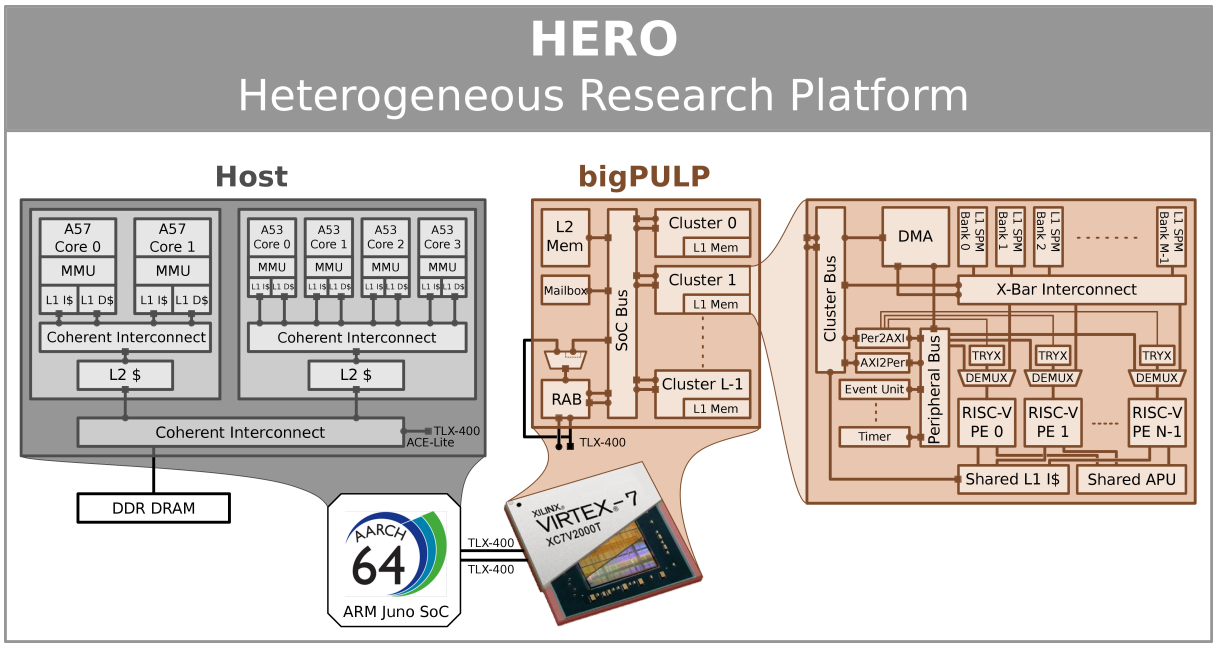

FIGURE 11. Block diagram of the a Hero SoC [55].

proves event latencies, interrupt lock times, and memory footprint at a moderate cost of Field-programmable Gate Array (FPGA) resources.

Malenko and Baunach [36] employ a microkernel-based operating system named SmartOS, which seeks to provide basic functionality in privilege-mode while other functionalities run in user-mode. Further, to evaluate the reliability of a RISC-V-based SoC, [58] compares five algorithms against a Linux-based OS and a bare-metal implementation.

3) Development Tools

The work of [82], illustrated in Figure 15, proposes a design method to generate domain-specific many-core architectures with provided frameworks and automated steps using software tools. The solution facilitates engineering and creates many-core architectures with different configurations, including core augmentation through instruction extensions and custom accelerators.

Torres-Snchez et al. [66] evaluate the development environment toolchains and debugging process concerning the Sipeed MAIX Go development board and Tiny YOLO v2 support by deploying a low-power IoT edge application, achieving good performance and cost characteristics.

Herdt and Drechsler's [62] proposal provides an automated formal verification tailored for SystemC-based VP on top of a RISC-V ISA and significantly improves verification quality and reducing overall verification effort.

Furthermore, [60] provides a library to run lightweight and energy-efficient neural networks. Further, the framework automates deployments on MCUs with and without an FPU. Finally, [83] proposal enables ISA designers to iteratively refine and evaluate ISA specifications, allowing one to improve upon each result.

B. OPERATING SYSTEM ARCHITECTURE

With the growing complexity of computer hardware, the operating system provides an abstraction layer to the user, acting as an interface between applications and computer hardware, enabling programmers to develop software without knowing much of the underlying details and executing software efficiently. The ISA defines the OS capabilities, as it defines the available machines' instructions to the application. RISC-V has a defined subset of OS support as well as user and machine instructions [73], [84].

1) Memory Model

The work of [82] uses scratchpad memory to avoid having to deal with cache coherence issues. Further, all cores and components in the architecture share the same address space. The memory module routes memory accesses to the data cache or another component through the crossbar network.

The SmartOS employed by [36] organizes the memory with the linker script to structure the Task Control Block (TCB), Resource Control Blocks (RCB), and Event Control Blocks (ECB) arrangements and the regions for task stack, data, and entry function.

The cache controller of NOEL-V supports a store buffer First In, First Out (FIFO) with one cycle per store and a wide Advanced High-performance Bus (AHB) data width support to enable start stores and fast cache refill [19].

Trippel et al. [83] propose a memory verification model to check for bugs on hardware and software memory models by providing a tool capable of verifying High-level Language (HLL), compilers, and ISAs uphold MCM requirements.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3174125, IEEE Access

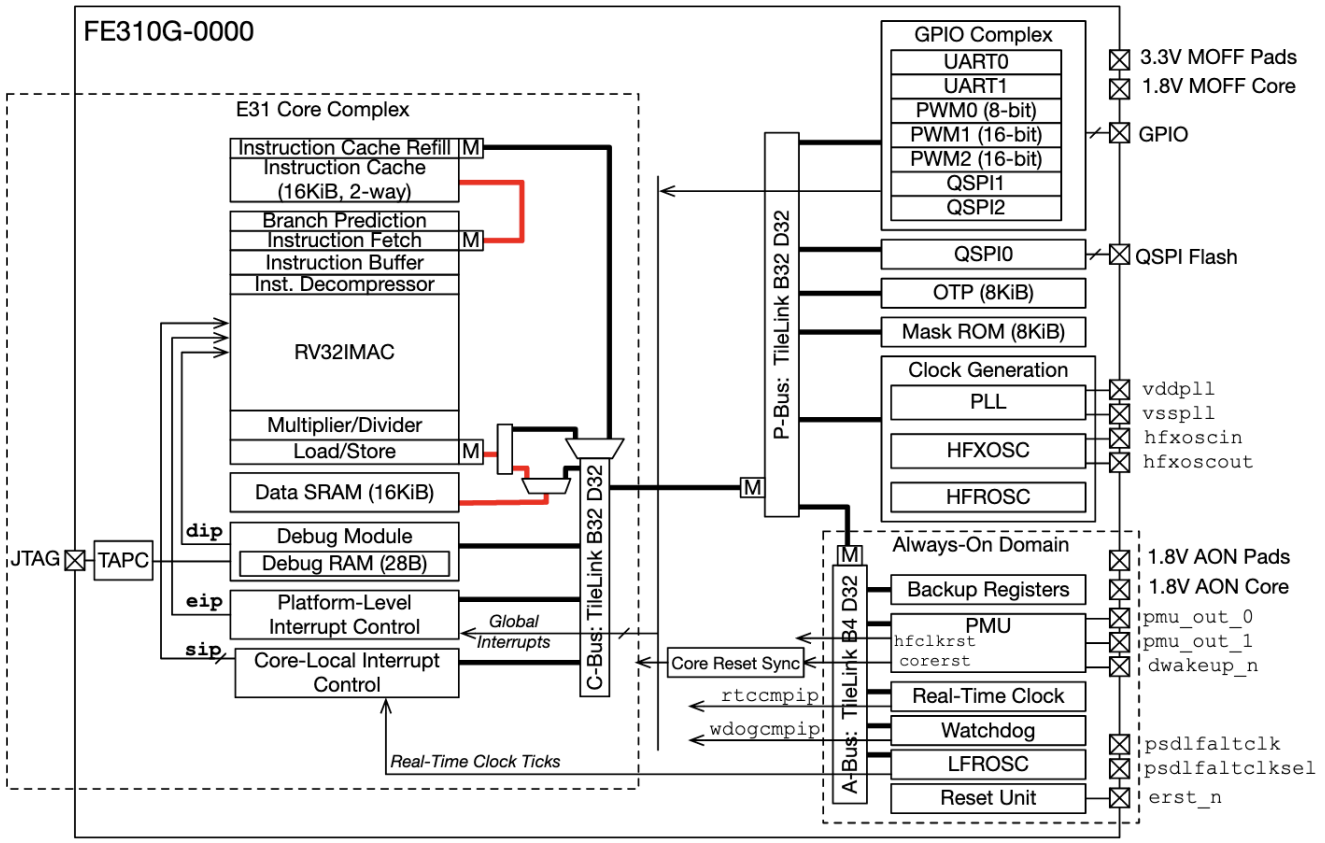

FIGURE 12. Block diagram of the FE3100-G00 SoC [61].

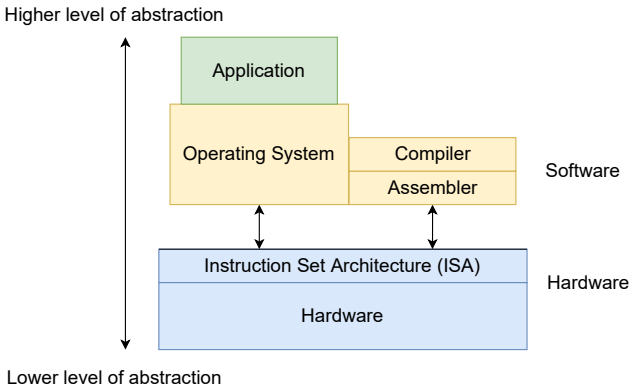

FIGURE 13. Simplified computer abstraction.

Further, the authors uncovers potential inefficiencies of the RISC-V ISA specification and identifies possible solutions to mitigate these inefficiencies.

2) Scheduler support

A processor scheduler assigns processes to be executed by the processor at a given time, meeting system response time, throughput, and processor efficiency objectives and allowing multiple processes to exist concurrently in a multiprogramming system. Furthermore, schedulers can also have real-time features, where it guarantees that high-priority tasks execute within a specific time constraint, and is characterized as a failure if the execution does not meet the conditions.

The work of [85] implements an improved version of a deterministic coprocessor task scheduler based on the Earliest-Deadline First (EDF) algorithm. In addition, the work adds support for CPUs to run two or four real-time tasks in parallel, improving real-time system performance and reducing resource costs using the Heap Queue sorting architecture for the Ready Queue implementation. The improved coprocessor and the task scheduler were described in SystemVerilog and verified by simulations and a simpler version of the Universal Verification Methodology (UVM).

Naylor et al. [86] explore the potential of a distributed soft-processor overlay programmed in software to deliver good performance for FPGA clusters. The work compares a Xeon cluster against a 12-FPGA 12,288 RISC-V threads to demonstrate the use of hardware multithreading to achieve a fast, space-efficient, and high-throughput overlay. The work implements a barrel-scheduled multithreaded core that uses a large subset of the RV32IMF ISA. In addition, the work uses a FPGA-optimized hyperthreaded RISC-V soft-core named

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3174125, IEEE Access

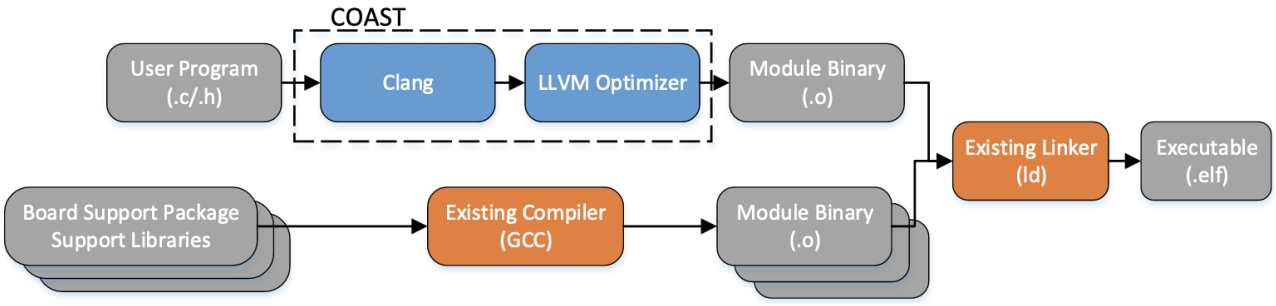

FIGURE 14. COAST proposal [75].

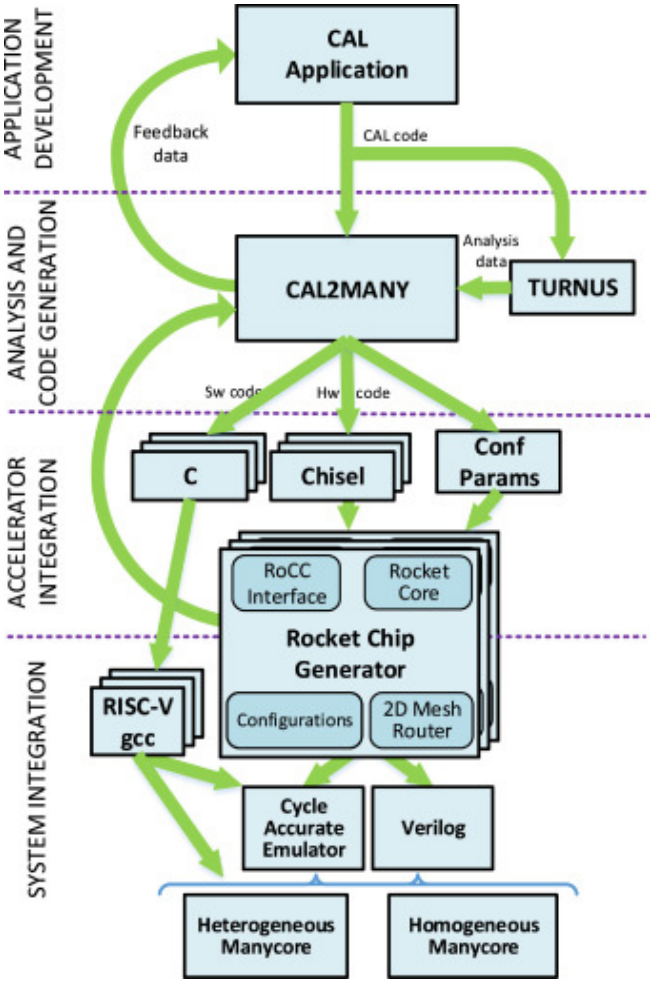

FIGURE 15. The realization of the design method including the tools and their inputs and outputs [82].

Tinsel, implementing a barrel-scheduled multithreaded core that uses a large subset of the RV32IMF ISA. Tinsel tolerates inherent latencies of a floating-point operation and off-chip memory access. Further, it provides a programming environment through an interface on top of the Tinsel Application Programming Interface (API), abstracting architectural details, and handling graph mapping onto the overlay.

Tasks in SmartOS, employed by [36], are preemptive with unique static priorities defined at compile-time and active priorities dynamically modified by the resource manager and used by the scheduler. Further, SmartOS's kernel uses a TCB structure for managing tasks.

3) Inter-Process Communication

Processes may need to communicate with one another, requiring the OS to provide a communication mechanism, preventing race conditions and the proper sequence of communications. If process B requires data from process A, process A needs to run before process B. The communication should be well-structured and preferably without using interrupts. The communication the OS provides is known as IPC [73], [84].

The work of [64] seeks to provide architectural support for a secure and efficient cross-process. The work, illustrated in Figure 16, proposes a hardware-assisted OS primitive named Cross Process Call (XPC), which uses an asynchronous IPC across different address spaces and enables a direct switch between IPC caller and callee without trapping into the kernel. XPC improves throughput using a new address-space mapping mechanism named `relay-seg` and provides a multithreading API with the migration thread model. The work supports split thread state, per-invocation C-Stack, and Android's Ashmem subsystem. The authors assessed performance, achieving a 0.3ms latency for 4KB data, with a 1.6x improvement, mainly from the secure zero-copying message transfer. XPC is compatible with traditional address-space isolation and can easily integrate with existing OS kernels.

Further, the work of [87] proposes in-process isolation based on dynamic memory protection domains. When required, the work uses a shared memory when sharing data with different domains, enforcing security with a set of protection keys and the associated memory, usage rights, and their allowed entry points by domains.

4) Network Stack

The Sipeed MAIX-I employed by [66] features a highly integrated Espressif ESP8285 SoC with complete self-contained Wi-Fi networking capabilities. Du et al. [64] use the lwIP network stack as a network stack server for the microkernel

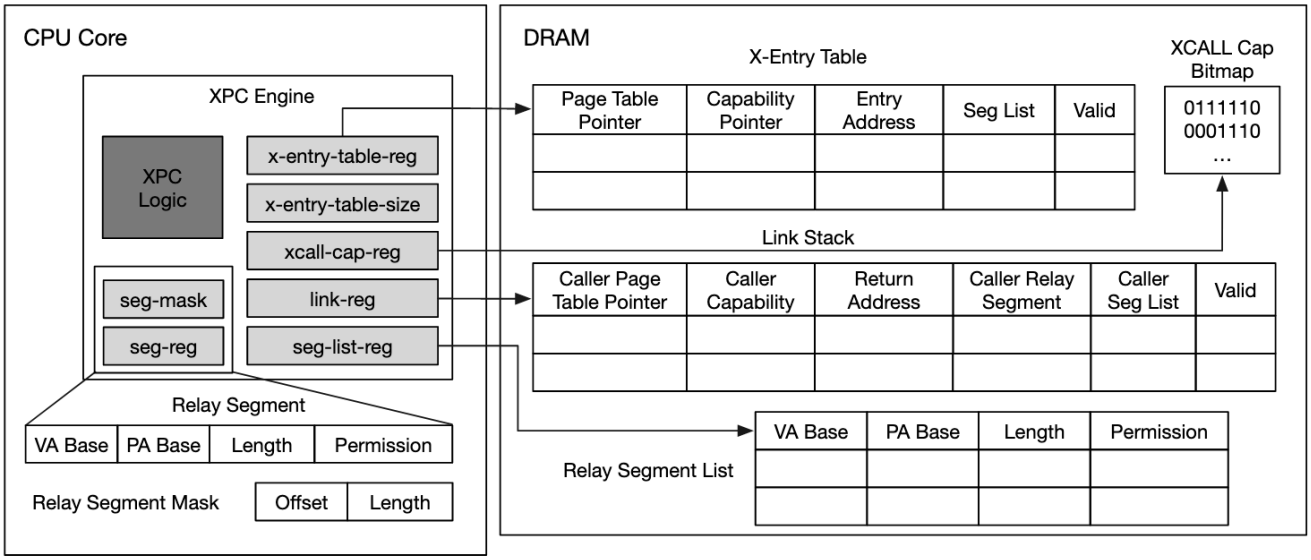

FIGURE 16. XPC architecture [64].

and a loopback device driver to pack and send to the network device server. Further, the proposal of SeRot uses the LKL network stack to provide network functionalities to the enclaved application [88].

5) File system support

Few developments have been reported concerning the file system support in operating systems for RISC-V. In this context, we can cite the work of [88], whose proposal enables enclaved applications to access the file system by using the Linux Kernel Library (LKL) for accessing and managing file system calls. Further, SeRot implements an ext4 file system based on LKL.

C. DISCUSSION

With little time in the market, RISC-V's software ecosystem includes compilers, state-of-the-art OSs, tools to generate domain-specific architectures, and development flow to ease software/hardware interfacing. In addition, the open specification of RISC-V has made it possible for one to support and implement robust software architectures to meet a range of requirements.

With the increasing demand for computing power, the simplicity of the RISC-V ISA enables OS engineers to support the architecture and contribute to the RISC-V specification effortlessly. On the other hand, the growing complexity of architectures such as x86 ISA makes it difficult for developers to keep up and contribute to the development of the ISA specification.

RISC-V's openness has led to a wide range of RISC-V applications, including academic and industrial usage, helping researchers explore the benefits of an open ISA for software development. Overall, the works presented have shown how RISC-V can support software and operating systems with sophisticated IPC, memory models, compiler techniques, and toolchains.

VII. DEPLOYMENT FEATURES

To further investigate the RISC-V software ecosystem, this section reviews the works that tackle security, reliability, and power management characteristics of RISC-V in order to improve the executing software.

A. SECURITY

Fault injections can be used to measure coverage, latency parameters, explore error propagation, and analyze how the system's workload and fault handling capabilities. Further, different fault injections models can be used to exploit security characteristics of given hardware [89], requiring countermeasures to avoid and protect such exploits from happening.

Laurent et al. [90] explore fault injection at the microarchitectural layer and propose a cross-layer approach. This approach cojoins software and hardware characteristics to improve countermeasures with reasonable overhead. The authors have shown faulty behaviors in a RISC-V processor by analyzing the RTL simulation, the consequences on software, and architectural aspects concerning security. In the experiments, the authors inject single-bit faults to simulate faulty behaviors in a LowRISC processor. The simulation results of fault injection have shown behaviors of forwarding capabilities, speculative execution, and writing in a General-Purpose Register (GPR) during branch instruction. The authors also review software fault models and provide alternatives to bypass the countermeasures set by data- and control-flow integrity. Further, the work considers fault attacks that affect control signals in the pipeline and ensures no exception gets raised by the processor during the analysis of the faulty behavior.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3174125, IEEE Access

Similarly, in another work [41], the same authors explore how software and hardware characteristics can lead to successful attacks. They provide countermeasures against hidden registers attacks by exploring the processor's multiplier unit and forwarding characteristics, which are invisible from the software point of view and are entirely dependent on the processor implementation.

In the context of coverage analysis and fault simulation, the work by [91] introduces a metric for RISC-V instructions and registers coverage for binary software to measure if the instructions are executed, and GPRs, CSRs, and FPRs are accessed. The work further analyzes and compares three available RISC-V test suites and combines them to a unified test suite, reaching 100% GPR coverage.

Hunt et al. [92] argue that hardware enforcement isolation mechanisms have historically been the software system designers' basis for a secure system. However, providing redundant protection for isolation by combining software and hardware techniques can improve system security. Zhang et al. [79] provide a kernel isolation method by implementing a Kernel Page-Table Isolation (KPTI) with a channel for accessing userspace from the kernel space by a block of a shared memory region. Further, the framework reduces the impact of human error during the development phase. Similarly, [36] explore memory isolation techniques and implement a hardware/software co-design approach for memory isolation by providing two hardware components: Device Driver Isolation Module (DDIM) and System Call Tracing Module (SCTM). Authorization of a system call happens after the decode stage of the pipeline.

Hwang et al. [42] introduce a new hardware-based monitoring platform to ensure kernel integrity from the outside host system named RiskiM and an interface architecture named PEMI. In order to overcome the semantic gap issue, PEMI provides all internal states of the host system to RiskiM to fulfill its monitoring task and protect the kernel in the presence of attacks. Furthermore, [43] proposes a hardware design protection against hardware Trojans inserted during the production phase through netlist obfuscation provided by logic locking.

In [56], Fischer et al. exploit hardware misconfigured access control of Read-Only Memory (ROM) a well as incorrect implementations of SoC firmware. Further, the work explores memory overwrites of peripheral regions and has found no code that implements the Counter (CTR) block cipher mode.

Auer et al. [31] address three high-level goals identified by the NISTIR 8228 report to provide a secure architecture on IoT devices. The work uses a secure-boot mechanism with a certificate chain to authenticate boot images, and update verification happens during runtime. Furthermore, [52] provide a RISC-V system compatible with TEEs and featuring a security algorithm accelerator. The work features SHA-3 and Ed25519 accelerators and provides a procedure for the software, composed of a root of trust to authenticate a Linux bootloader.

Liu et al. [88] propose a secure runtime system named SeRot, illustrated in Figure 17, which addresses TEE primitives to support unmodified applications. Further, the work uses LKL to allow external calls to the host machine and provides API and ABI protection.

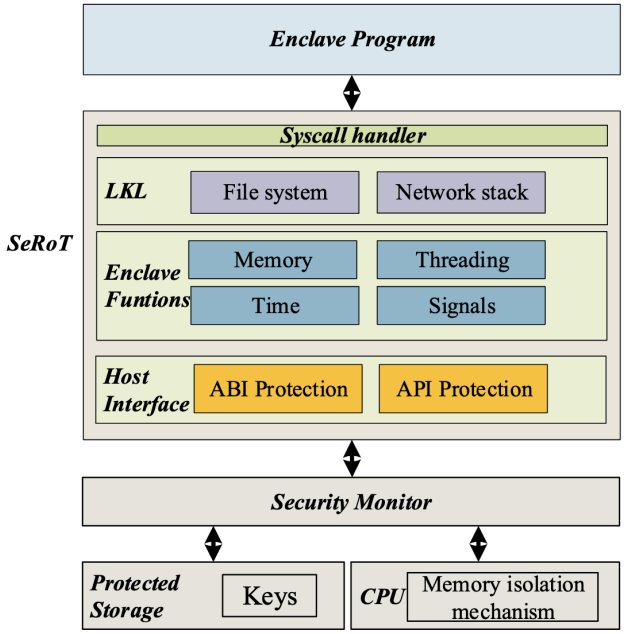

FIGURE 17. Architecture of SeRot [88].

In addition, [59] provides a solution for securing the processor by implementing a self-authenticated secure boot and provide Information Flow Tracking (IFT) to detect and stop memory corruption attacks.

Dessouky et al. [57] provide a testbed of real-world software-exploitable RTL bugs based on RISC-V SoCs, exploring TLB, cache, and memory attacks to identify specific vulnerabilities classes.

Finally, [93] proposes a Runtime Scope Enforcement (RSE) approach to mitigate all known Data-oriented Programming (DOP) attacks efficiently on memory-unsafe programming languages. The proposed technique enforces memory safety constraints during compile-time, resulting in a low-performance overhead.

B. RELIABILITY

With difficulty in ensuring the trustworthiness of the fabrication process of silicon devices, [94] propose an architectural protection mechanism, shown in Figure 18, to detect hardware trojans infesting the instruction and data memories of the system by shielding the communication of the processor and memory in a microprocessor-based system. The proposal relies on Bloom Filters (BF), which guarantees no false alarms and a small configurable percentage of undetected alarms, resulting in a 99% detection rate of possible hardware trojans activation.

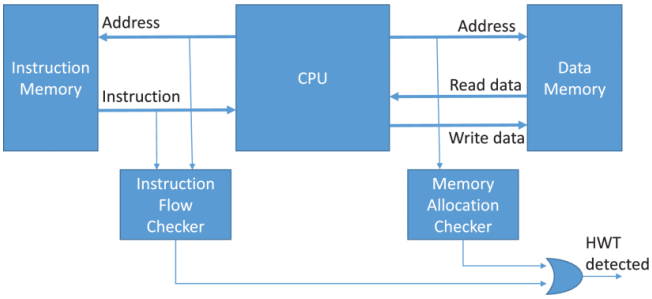

FIGURE 18. The protection-based architecture proposed by [94].

Similarly, [95] investigates undocumented instructions on RISC-V and ARM ISAs and propose two methods to look for undocumented instructions. Both methods execute a single instruction in a controlled manner, allowing the processor to determine if the instruction word is valid by comparing the results to the processor ISA specification.

Further, [96] seeks to provide an efficient approach to fault effect analysis and simulation by introducing a virtual prototype-based approach. By considering permanent and transient faults, single n-bitflips are injected into the fetched/execute instructions, GPRs and CSRs of a RISC-V processor model and simulated to analyze the impact during software execution. In contrast, [58] found that SEUs barely affect the number of silent data corruption in the presence of an OS. In addition, the single event functional interrupt rate increases about 3.6 times compared to the bare-metal program execution. Similarly, [97] evaluate the reliability of software applications running on soft microprocessors against SEUs that affect configuration and microprocessor memories. The work compares the utilization of hardened-by-replication software with the baseline software by testing a set of benchmarks.

Liu et al. [34] propose a domain-specific architecture design for CNN-based Artificial Intelligence (AI) IoT application, among with a heterogeneous processor design and accelerator for the inference of CNNs. Along the lines of domain-specific architectures, [82] generate domain-specific manycore architectures with the provided framework and automated steps with software tools.

In [93], Nyman et al. extend the ISA with seven new Storage Region Stack (SRS) management instructions to provide context-specific enforcement. Further, the authors modify the instruction decode stage of the processor pipeline to interpret the new instructions. Similarly, [59] modifies the execute stage of the processor pipeline to add support to tag propagation and tag check features.

To introduce a non-intrusive approach to optimize FPU bit-width for approximate computing, [98] proposes a methodology to help designers select the most optimized FPU configuration that meets a given Quality of Results (QoR) threshold for a specific application. In addition, [63] presents AxSWGenfor, an automated quality-driven methodology that combines different approximate computing techniques to explore and apply to error-tolerant sections of applications.

C. LOW-POWER

Imianosky et al. [99] evaluate CCSDS 123 Compressor performance and power consumption on RISC-V and ARM architectures by executing the algorithms in two OS with RISC-V support: FreeRTOS and Zephyr. The authors concluded that RISC-V uses less power when compared to the ARM processor, while ARM offers higher performance and lower energy consumption than RISC-V. The work also showed FreeRTOS has a lower overhead to the algorithm execution in comparison to Zephyr when executed on a RISC-V processor.

The SoC adopted by [31] has low-power characteristics to a low of 0.4V, enabling designers to perform power-tradeoffs at runtime by adjusting the back-gate bias voltage. In contrast, [62] proposes methods of improving verification quality to aid decisions that significantly impact the power consumption strategies by validating firmware-based power management implementations.

The proposal of [100] is well-suited for low-intensity tasks at low data rates and uses a microcontroller working at a low operating frequency of 25MHz. Further, the work presents ultra-low-power comparator circuits, which run on low frequencies to improve the power overshoot identification. Similar, the work of [66] keeps the CPU below 350mW when running face-detection routines and 35mW with both cores when the CPU is waiting for interrupts.

The methodology proposed by [63] seeks to minimize energy consumption and meets the defined threshold for a given application-level quality metric. Furthermore, the toolkit proposed by [60] takes multi-layer perceptrons trained with FANN to generate code for energy-efficient neural networks on microcontrollers.

D. DISCUSSION

This section sought to review works that tackled or have similarities with security, reliability, and low-power characteristics. The RISC-V's features enable designers to implement novel resources to improve the architecture's security and reliability. Several works took advantage of the open ISA in order to analyze its implementation and test the security, reliability, and power efficiency.

Due to RISC-V's nature of supporting a variety of environments, a set of works have focused on providing TEE techniques in order to secure processes as well as to run applications in an environment with low-power requirements.

VIII. CONCLUSION

Despite the novelty of RISC-V, a large variety of works aims at adapting and using RISC-V architectures to explore the capabilities of an open ISA. The different implementations of RISC-V aspire to improve or extend the basic specifications to satisfy peculiar computing needs. Low-level programming environments strive for a stable ISA to strongly comply with

high-level programming environments. RISC-V's specification has primarily fulfilled these environments by providing engineers and researchers with a reliable frozen ISA base. The RISC-V's goal of supporting a broad set of computing environments has allowed enterprises and academia to tackle specific software requirements with low cost and flexibility.

This survey demonstrated how RISC-V is prevalent in works that seek to tackle specific requirements and how the community can benefit from an open ISA specification. Furthermore, given RISC-V's novelty and the rich software ecosystem, the community has well received and contributed to adopting RISC-V architectures into a wide range of systems from small and constrained devices to large-scale computers.

REFERENCES

[1] A. Waterman, Y. Lee, D. A. Patterson, and K. Asanović, "The risc-v instruction set manual, volume i: Base user-level isa," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2011-62, May 2011. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2011/EECS-2011-62.html

[2] D. Patterson and A. Waterman, *The RISC-V Reader: An Open Architecture Atlas*, 1st ed. California: Strawberry Canyon, 2017.

[3] T. Chen and D. A. Patterson, "Risc-v geneology," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-6, Jan 2016. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-6.html

[4] G. S. Nicholas, Y. Gui, and F. Saqib, "A survey and analysis on soc platform security in arm, intel and risc-v architecture," in *2020 IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS)*, IEEE. Springfield, MA, USA: IEEE, 2020, pp. 718–721.

[5] T. Lu, "A survey on RISC-V security: Hardware and architecture," *CoRR*, vol. abs/2107.04175, 2021. [Online]. Available: https://arxiv.org/abs/2107.04175

[6] R. Newell, J. Xie, and H. Handschuh, "Survey of notable security-enhancing activities in the risc-v universe," in *17th International Workshop on Cryptographic Architectures Embedded in Logic Devices, CryptArchi*. Prague, Czech Republic: CryptArchi, 2019. [Online]. Available: https://labh-curien.univ-st-etienne.fr/cryptarchi/workshop19/abstracts/newell.pdf

[7] B. Marshall, G. R. Newell, D. Page, M.-J. O. Saarinen, and C. Wolf, "The design of scalar aes instruction set extensions for risc-v," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2021, no. 1, p. 109–136, Dec. 2020. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/8729

[8] A. Dörflinger, M. Albers, B. Kleinbeck, Y. Guan, H. Michalik, R. Klink, C. Blochwitz, A. Nechi, and M. Berekovic, "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

[9] I. Elsadek and E. Y. Tawfik, "Risc-v resource-constrained cores: A survey and energy comparison," in *2021 19th IEEE International New Circuits and Systems Conference (NEWCAS)*. Toulon, France: IEEE, 2021, pp. 1–5.

[10] A. Waterman, Y. Lee, D. A. Patterson, and K. Asanović, "The risc-v instruction set manual, volume i: User-level isa, version 2.2," EECS Department, University of California, Berkeley, Tech. Rep., May 2017. [Online]. Available: https://riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf

[11] ——, "The risc-v instruction set manual, volume i: User-level isa, version 2.0," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2014-54, May 2014. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-54.html

[12] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design RISC-V Edition: The Hardware Software Interface*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2017.

[13] A. Waterman, Y. Lee, R. Avizienis, D. A. Patterson, and K. Asanović, "The risc-v instruction set manual volume ii: Privileged architecture version 1.9," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-129, Jul 2016. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-129.html

[14] M. Larabel. (2016) Nvidia is building its next-gen falcon controller using risc-v - phoronix. Phoronix. [Online]. Available: https://www.phoronix.com/scan.php?page=news_item&px=NVIDIA-RISC-V-Next-Gen-Falcon

[15] A. Shilov. (2021) Western digital reveals swerv risc-v core, cache coherency over ethernet initiative. AnandTech. [Online]. Available: https://www.anandtech.com/show/13678/western-digital-reveals-swerv-risc-v-core-and-omnixtend-coherency-tech

[16] J. Hsu, "Risc-v star rises among chip developers worldwide - ieee spectrum," 2021. [Online]. Available: https://spectrum.ieee.org/tech-talk/semiconductors/design/riscv-rises-among-chip-developers-worldwide

[17] C. Chen, X. Xiang, C. Liu, Y. Shang, R. Guo, D. Liu, Y. Lu, Z. Hao, J. Luo, Z. Chen, C. Li, Y. Pu, J. Meng, X. Yan, Y. Xie, and X. Qi, "Xuantie-910: A commercial multi-core 12-stage pipeline out-of-order 64-bit high performance risc-v processor with vector extension : Industrial product," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. Spain: IEEE, 2020, pp. 52–64.

[18] A. Shilov. (2021) Huawei's hisilicon develops first risc-v design to overcome arm restrictions tom's hardware. Tom's Hardware. [Online]. Available: https://www.tomshardware.com/news/huaweis-hisilicon-develops-first-risc-v-design-to-overcome-arm-restrictions

[19] J. Andersson, "Development of a noel-v risc-v soc targeting space applications," in *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. Los Alamitos, CA, USA: IEEE Computer Society, jul 2020, pp. 66–67. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/DSN-W50199.2020.00020

[20] S. Shankland. (2021) Apple shows interest in risc-v chips, a competitor to iphones' arm tech. CNet. [Online]. Available: https://www.cnet.com/tech/mobile/apple-shows-interest-in-risc-v-chips-a-competitor-to-iphones-arm-tech/

[21] 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.

[22] P. platform. (2021) Pulp users. Pulp-Platform. [Online]. Available: https://www.pulp-platform.org/pulp_users.html

[23] J. Osier Mixon, "Semico forecasts strong growth for risc-v," November 2019. [Online]. Available: https://riscv.org/announcements/2019/11/9679/

[24] S. R. . C. group. (2019) Risc-v market analysis: The new kid on the block semico research. Semico. [Online]. Available: https://semico.com/content/risc-v-market-analysis-new-kid-block

[25] F. Gómez, M. Masmano, V. Nicolau, J. Andersson, J. Le Rhun, D. Trilla, F. Gallego, G. Cabo, and J. Abella Ferrer, "De-risc – dependable real-time infrastructure for safety-critical computer systems," *Ada User Journal*, vol. 41, no. 2, p. 107–112, Jun 2020. [Online]. Available: http://hdl.handle.net/2117/341317

[26] C. Hernàndez, J. Flieh, R. Paredes, C.-A. Lefebvre, I. Allende, J. Abella, D. Trillin, M. Matschnig, B. Fischer, K. Schwarz, J. Kiszka, M. Rönnbäck, J. Klockars, N. McGuire, F. Rammerstorfer, C. Schwarzl, F. Wartet, D. Lüdemann, and M. Labayen, "Selene: Self-monitored dependable platform for high-performance safety-critical systems," in *2020 23rd Euromicro Conference on Digital System Design (DSD)*. Kranj, Slovenia: IEEE, 2020, pp. 370–377.

[27] S. Di Mascio, A. Menicucci, G. Furano, C. Monteleone, and M. Ottavi, "The case for risc-v in space," in *Applications in Electronics Pervading Industry, Environment and Society*, S. Saponara and A. De Gloria, Eds. Cham: Springer International Publishing, 2019, pp. 319–325.

[28] G. V. Research. (2021, Jul) Industrial internet of things market worth $1.11 trillion by 2028. Grand View Research. [Online]. Available: https://www.grandviewresearch.com/press-release/global-industrial-internet-of-things-iiot-market

[29] ——. (2021, Jun) Industrial internet of things market size report. Grand View Research. [Online]. Available: https://www.grandviewresearch.com/industry-analysis/industrial-internet-of-things-iiot-market

[30] K. Boeckl, K. Boeckl, M. Fagan, W. Fisher, N. Lefkovitz, K. N. Megas, E. Nadeau, D. G. O'Rourke, B. Piccarreta, and K. Scarfone, *Considerations for managing Internet of Things (IoT) cybersecurity and privacy risks*. Gaithersburg, Maryland, United States: US Department of Commerce, National Institute of Standards and Technology, 2019.

[31] L. Auer, C. Skubich, and M. Hiller, "A security architecture for risc-v based iot devices," in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*. Florence, Italy: IEEE, 2019, pp. 1154–1159.

[32] M. Malenko and M. Baunach, "Hardware/software co-designed peripheral protection in embedded devices," in *2019 IEEE International Conference on Industrial Cyber Physical Systems (ICPS)*. Taipei, Taiwan: IEEE, 2019, pp. 790–795.

[33] V. B. Y. Kumar, N. Gupta, A. Chattopadhyay, M. Kasper, C. Krauß, and R. Niederhagen, "Post-quantum secure boot," in *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*. Grenoble, France: IEEE, 2020, pp. 1582–1585.

[34] Z. Liu, J. Jiang, G. Lei, K. Chen, B. Qin, and X. Zhao, "A heterogeneous processor design for cnn-based ai applications on iot devices," *Procedia Computer Science*, vol. 174, pp. 2–8, 2020, 2019 International Conference on Identification, Information and Knowledge in the Internet of Things. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050920315611

[35] SiFive. (2021) Sifive intelligence. SiFive. [Online]. Available: https://www.sifive.com/cores/intelligence

[36] M. Malenko and M. Baunach, "Device driver and system call isolation in embedded devices," in *2019 22nd Euromicro Conference on Digital System Design (DSD)*. Kallithea, Greece: IEEE, 2019, pp. 283–290.

[37] *PULPino: Datasheet*, PULP Platform, 6 2017. [Online]. Available: https://pulp-platform.org/docs/pulpino_datasheet.pdf

[38] e. a. lowRISC, "Ibex risc-v core," https://github.com/lowRISC/ibex, 2021.

[39] e. OpenHW Group, "Openhw group core-v cv32e40p risc-v ip," https://github.com/openhwgroup/cv32e40p, 2021.

[40] 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.

[41] J. Laurent, V. Beroulle, C. Deleuze, and F. Pebay-Peyroula, "Fault injection on hidden registers in a risc-v rocket processor and software countermeasures," in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*. Florence, Italy: IEEE, 2019, pp. 252–255.

[42] D. Hwang, M. Yang, S. Jeon, Y. Lee, D. Kwon, and Y. Paek, "Riskim: Toward complete kernel protection with hardware support," in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*. Florence, Italy: IEEE, 2019, pp. 740–745.

[43] D. Šišejković, F. Merchant, L. M. Reimann, R. Leupers, M. Giacometti, and S. Kegreiß, "A secure hardware-software solution based on risc-v, logic locking and microkernel," in *Proceedings of the 23th International Workshop on Software and Compilers for Embedded Systems*, ser. SCOPES '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 62–65. [Online]. Available: https://doi.org/10.1145/3378678.3391886

[44] P. Bardonek and M. Zachariášová, "Using control logic drivers for automated generation of system-level portable models," in *2020 23rd International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*. Novi Sad, Serbia: IEEE, 2020, pp. 1–4.

[45] J. Zhao, B. Korpan, A. Gonzalez, and K. Asanovic, "Sonicboom: The 3rd generation berkeley out-of-order machine," p. 7, May 2020.

[46] BOOM, "The berkeley out-of-order risc-v processor," https://github.com/riscv-boom/riscv-boom, 2021.

[47] J. Lee, H. Chen, J. Young, and H. Kim, "Risc-v fpga platform toward ros-based robotics application," in *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*. Gothenburg, Sweden: IEEE, 2020, pp. 370–370.

[48] SiFive. (2021) Sifive performance p550 core sets new standard as highest performance risc-v processor ip - sifive. SiFive. [Online]. Available: https://www.sifive.com/press/sifive-performance-p550-core-sets-new-standard-as-highest

[49] M. Larabel. (2021) Sifive announces the performance p550 as the fastest risc-v processor yet. Phoronix. [Online]. Available: https://www.phoronix.com/scan.php?page=news_item&px=SiFive-Performance-P550-P270

[50] Bloomberg. (2021) Sifive performance p550 core sets new standard as highest performance risc-v processor ip - bloomberg. Bloomberg. [Online]. Available: https://www.bloomberg.com/press-releases/2021-06-22/sifive-performance-p550-core-sets-new-standard-as-highest-performance-risc-v-processor-ip

[51] C. Gaisler. (2021) Noel-v processor. Gaisler. [Online]. Available: https://www.gaisler.com/index.php/products/processors/noel-v

[52] T.-T. Hoang, C. Duran, D.-T. Nguyen-Hoang, D.-H. Le, A. Tsukamoto, K. Suzaki, and C.-K. Pham, "Quick boot of trusted execution environment with hardware accelerators," *IEEE Access*, vol. 8, pp. 74 015–74 023, 2020.

[53] *PULPissimo: Datasheet*, PULP Platform, 3 2021. [Online]. Available: https://raw.githubusercontent.com/pulp-platform/pulpissimo/master/doc/datasheet/datasheet.pdf

[54] *Pulp Hardware Reference Manual*, PULP Platform, 2 2019. [Online]. Available: https://raw.githubusercontent.com/pulp-platform/pulp/master/doc/datasheet.pdf

[55] *HERO Documentation*, PULP Platform, 2021. [Online]. Available: https://pulp-platform.org/hero.html

[56] M. Fischer, F. Langer, J. Mono, C. Nasenberg, and N. Albartus, "Hardware penetration testing knocks your socs off," *IEEE Design Test*, vol. 38, no. 1, pp. 14–21, 2021.

[57] G. Dessouky, D. Gens, P. Haney, G. Persyn, A. Kanuparthi, H. Khattri, J. M. Fung, A.-R. Sadeghi, and J. Rajendran, "Hardfails: Insights into software-exploitable hardware bugs," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 213–230. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/dessouky

[58] I. Wali, A. Sánchez-Macián, A. Ramos, and J. A. Maestro, "Analyzing the impact of the operating system on the reliability of a risc-v fpga implementation," in *2020 27th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. Glasgow, UK: IEEE, 2020, pp. 1–4.

[59] A. S. Siddiqui, G. Shirley, S. Bendre, G. Bhagwat, J. Plusquellic, and F. Saqib, "Secure design flow of fpga based risc-v implementation," in *2019 IEEE 4th International Verification and Security Workshop (IVSW)*. Rhodes, Greece: IEEE, 2019, pp. 37–42.

[60] X. Wang, M. Magno, L. Cavigelli, and L. Benini, "Fann-on-mcu: An open-source toolkit for energy-efficient neural network inference at the edge of the internet of things," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4403–4417, 2020.

[61] *SiFive FE310-G000 Manual*, SiFive, 10 2017. [Online]. Available: https://static.dev.sifive.com/FE310-G000.pdf

[62] V. Herdt and R. Drechsler, "Efficient techniques to strongly enhance the virtual prototype based design flow," in *2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. Limassol, Cyprus: IEEE, 2020, pp. 182–187.

[63] J. Castro-Godínez, M. Shafique, and J. Henkel, "Towards quality-driven approximate software generation for accurate hardware: Work-in-progress," in *2020 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES)*. Shanghai, China: IEEE, 2020, pp. 12–14.

[64] D. Du, Z. Hua, Y. Xia, B. Zang, and H. Chen, "Xpc: Architectural support for secure and efficient cross process call," in *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*. Phoenix, AZ, USA: IEEE, 2019, pp. 671–684.

[65] *PolarFire SoC Advanced Datasheet*, Microsemi, 12 2019. [Online]. Available: https://www.microsemi.com/document-portal/doc_download/1244583-polarfire-soc-advance-datasheet

[66] E. Torres-Sánchez, J. Alastruey-Benedé, and E. Torres-Moreno, "Developing an ai iot application with open software on a risc-v soc," in *2020 XXXV Conference on Design of Circuits and Integrated Systems (DCIS)*. Segovia, Spain: IEEE, 2020, pp. 1–6.

[67] *RISC-V System emulator*, QEMU, 2021, revision: 703e8cd6. [Online]. Available: https://qemu.readthedocs.io/en/latest/system/target-riscv.html

[68] *About QEMU*, QEMU, 2021, revision: 6df743dc31a6a0b618042da2b550993c6e9767d1. [Online]. Available: https://qemu-project.gitlab.io/qemu/about/index.html

[69] *About*, Gem5, 2021. [Online]. Available: https://www.gem5.org/about

[70] *Spike RISC-V ISA Simulator*, RISC-V, 3 2021, revision: a31184c3de3fd981b4733b10554215db0e5aa85f. [Online]. Available: https://github.com/riscv/riscv-isa-sim

[71] B. Landers, "Rars - risc-v assembler and runtime simulator," https://github.com/TheThirdOne/rars, 2021.

[72] R. Giorgi and G. Mariotti, "Webrisc-v: A web-based education-oriented risc-v pipeline simulation environment," in *Proceedings of the Workshop*

on Computer Architecture Education, ser. WCAE'19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3338698.3338894

[73] W. Stallings, *Operating Systems: Internals and Design Principles, 9/e*, 9th ed. Indianapolis, Indiana, USA: Pearson IT Certification, 2018.

[74] G. Coulouris, J. Dollimore, T. Kindberg, and G. Blair, *Distributed Systems: Concepts and Design*, 5th ed. USA: Addison-Wesley Publishing Company, 2011.

[75] B. James, H. Quinn, M. Wirthlin, and J. Goeders, "Applying compiler-automated software fault tolerance to multiple processor platforms," *IEEE Transactions on Nuclear Science*, vol. 67, no. 1, pp. 321–327, 2020.

[76] M. Bohman, B. James, M. J. Wirthlin, H. Quinn, and J. Goeders, "Microcontroller compiler-assisted software fault tolerance," *IEEE Transactions on Nuclear Science*, vol. 66, no. 1, pp. 223–232, 2019.

[77] R. Stahl, D. Mueller-Gritschneder, and U. Schlichtmann, "Driver generation for iot nodes with optimization of the hardware/software interface," *IEEE Embedded Systems Letters*, vol. 12, no. 2, pp. 66–69, 2020.

[78] A. Fell, H. T. Pham, and S.-K. Lam, "Tad: Time side-channel attack defense of obfuscated source code," in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, ser. ASPDAC '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 58–63. [Online]. Available: https://doi.org/10.1145/3287624.3287694

[79] Q. Zhang, J. Qiao, Q. Meng, and Y. Chen, "Automatic kernel code synthesis and verification," *Computers & Security*, vol. 91, p. 101733, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404820300201

[80] M. N. Ince, J. Ledet, and M. Gunay, "Building an open source linux computing system on risc-v," in *2019 1st International Informatics and Software Engineering Conference (UBMYK)*. Ankara, Turkey: IEEE, 2019, pp. 1–4.

[81] C. Dietrich and D. Lohmann, "Osek-v: Application-specific rtos instantiation in hardware," *SIGPLAN Not.*, vol. 52, no. 5, p. 111–120, Jun. 2017. [Online]. Available: https://doi.org/10.1145/3140582.3081030

[82] S. Savas, Z. Ul-Abdin, and T. Nordström, "A framework to generate domain-specific manycore architectures from dataflow programs," *Microprocessors and Microsystems*, vol. 72, p. 102908, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0141933119302819

[83] C. Trippel, Y. A. Manerkar, D. Lustig, M. Pellauer, and M. Martonosi, "Tricheck: Memory model verification at the trisection of software, hardware, and isa," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 119–133. [Online]. Available: https://doi.org/10.1145/3037697.3037719

[84] A. S. Tanenbaum and H. Bos, *Modern Operating Systems*, 4th ed. USA: Prentice Hall Press, 2014.

[85] L. Kohútka and V. Stopjaková, "Novel efficient on-chip task scheduler for multi-core hard real-time systems," *Microprocessors and Microsystems*, vol. 76, p. 103083, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S014193311930585X

[86] M. Naylor, S. W. Moore, and D. Thomas, "Tinsel: A manythread overlay for fpga clusters," in *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*. Barcelona, Spain: IEEE, Sep. 2019, pp. 375–383.

[87] D. Schrammel, S. Weiser, S. Steinegger, M. Schwarzl, M. Schwarz, S. Mangard, and D. Gruss, "Donky: Domain keys – efficient in-process isolation for risc-v and x86," in *29th USENIX Security Symposium (USENIX Security 20)*. Boston, MA, USA: USENIX Association, Aug. 2020, pp. 1677–1694. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/schrammel

[88] J. Liu, Y. Qin, and D. Feng, "Serot: A secure runtime system on trusted execution environments," in *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. Guangzhou, China: IEEE, 2020, pp. 30–37.

[89] I. Koren and C. M. Krishna, *Fault-Tolerant Systems*, 2nd ed. San Francisco (CA): Morgan Kaufmann, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128181058000115

[90] J. Laurent, V. Beroulle, C. Deleuze, F. Pebay-Peyroula, and A. Papadimitriou, "Cross-layer analysis of software fault models and countermeasures against hardware fault attacks in a risc-v processor," *Microprocessors and Microsystems*, vol. 71, p. 102862, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0141933118304745

[91] P. Adelt, B. Koppelmann, W. Mueller, and C. Scheytt, "Register and instruction coverage analysis for different risc-v isa modules," in *MBMV 2021; 24th Workshop*. Online: VDE, 2021, pp. 1–8.

[92] T. Hunt, Z. Jia, V. Miller, C. J. Rossbach, and E. Witchel, "Isolation and beyond: Challenges for system security," in *Proceedings of the Workshop on Hot Topics in Operating Systems*, ser. HotOS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 96–104. [Online]. Available: https://doi.org/10.1145/3317550.3321427

[93] T. Nyman, G. Dessouky, S. Zeitouni, A. Lehikoinen, A. Paverd, N. Asokan, and A.-R. Sadeghi, "Hardscope: Hardening embedded systems against data-oriented attacks," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*. Las Vegas, NV, USA: IEEE, 2019, pp. 1–6.

[94] A. Bolat, L. Cassano, P. Reviriego, O. Ergin, and M. Ottavi, "A microprocessor protection architecture against hardware trojans in memories," in *2020 15th Design Technology of Integrated Systems in Nanoscale Era (DTIS)*. Marrakech, Morocco: IEEE, 2020, pp. 1–6.

[95] R. Dofferhoff, M. Göebel, K. Rietveld, and E. van der Kouwe, "iscanu: A portable scanner for undocumented instructions on risc processors," in *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. Valencia, Spain: IEEE, 2020, pp. 306–317.

[96] P. Adelt, B. Koppelmann, W. Mueller, and C. Scheytt, "A scalable platform for qemu based fault effect analysis for risc-v hardware architectures," in *MBMV 2020 - Methods and Description Languages for Modelling and Verification of Circuits and Systems; GMM/ITG/GI-Workshop*. Stuttgart, Germany: IEEE, 2020, pp. 1–8.

[97] C. De Sio, S. Azimi, A. Portaluri, and L. Sterpone, "Seu evaluation of hardened-by-replication software in risc- v soft processor," in *2021 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. Athens, Greece: IEEE, 2021, pp. 1–6.

[98] N. A. Said, M. Benabdenbi, and K. Morin-Allory, "Fpu bit-width optimization for approximate computing: A non-intrusive approach," in *2020 15th Design Technology of Integrated Systems in Nanoscale Era (DTIS)*. Marrakech, Morocco: IEEE, 2020, pp. 1–6.

[99] C. Imianosky, P. R. O. Valim, C. A. Zeferino, and F. Viel, "Evaluating the ccsds 123 compressor running on risc-v and arm architectures," in *2020 X Brazilian Symposium on Computing Systems Engineering (SBESC)*. Florianópolis, Brazil: IEEE, Nov 2020, pp. 1–7.

[100] I. Karageorgos, K. Sriram, J. Veselý, M. Wu, M. Powell, D. Borton, R. Manohar, and A. Bhattacharjee, "Hardware-software co-design for brain-computer interfaces," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. Valencia, Spain: IEEE, 2020, pp. 391–404.

BENJAMIN W. MEZGER received his B.S. in Computer Science (2019) from the University of Vale do Itajai, and is currently a M.S. student in Applied Computer Science at the University of Vale do Itajai. His topics of interest are RISC-V, Operating Systems, and Fault Tolerance.

DOUGLAS A. SANTOS received his B.E. in Computer Engineering (2019) and his M.S. in Applied Computer Science (2021) from the University of Vale do Itajai. He is currently a Ph.D. student at the Laboratoire d'Informatique, de Robotique et de Microelectronique de Montpellier (LIRMM). His topics of interest are RISC-V, Embedded Systems, and Fault Tolerance.

LUIGI DILILLO received the Diploma in Electronic Engineering from the Politecnico di Torino (Italy) in 2001. He next obtained his Ph.D. in microelectronics at the University of Montpellier. Currently, he is a CNRS researcher at the LIRMM laboratory. The fields of interest of his researches are Memory Test and Reliability, Power-Aware Test, Radiation Impact on Electronics, Radiation monitoring, Space and Radiation Hardened Systems Design.

CESAR A. ZEFERINO received his Ph.D. in Computer Science from the Federal University of Rio Grande do Sul, Brazil, in 2003. He is a Full Professor and the Director of the School of Sea, Science and Technology of the University of Vale do Itajai – Univali, Brazil. He also heads the Laboratory of Embedded and Distributed Systems of Univali. His topics of interest are Digital Systems Design, Embedded Systems, Networks-on-Chip, and Hardware Acceleration.

DOUGLAS R. MELO received his B.E. in Computer Engineering (2008) and his M.S. in Applied Computer Science (2012) from the University of Vale do Itajai, and his Ph.D. in Electrical Engineering (2020) from the Federal University of Santa Catarina. He is currently an Adjunct Professor at the University of Vale do Itajai and a Researcher at the Laboratory of Embedded and Distributed Systems. His topics of interest are Systems-on-Chip, Networks-on-Chip, and Fault Tolerance.