information

Article

A Survey on RISC-V-Based Machine Learning Ecosystem

Stavros Kalapothas [1], Manolis Galetakis [2], Georgios Flamis [1], Fotis Plessas [2] and Paris Kitsos [1,*]

[1] Electronic Circuits, Systems and Applications (ECSA) Laboratory, Electrical and Computer Engineering Department, University of Peloponnese, 26334 Patras, Greece
[2] Department of Electrical and Computer Engineering, University of Thessaly, 38221 Volos, Greece
* Correspondence: kitsos@uop.gr

Abstract: In recent years, the advancements in specialized hardware architectures have supported the industry and the research community to address the computation power needed for more enhanced and compute intensive artificial intelligence (AI) algorithms and applications that have already reached a substantial growth, such as in natural language processing (NLP) and computer vision (CV). The developments of open-source hardware (OSH) and the contribution towards the creation of hardware-based accelerators with implication mainly in machine learning (ML), has also been significant. In particular, the reduced instruction-set computer-five (RISC-V) open standard architecture has been widely adopted by a community of researchers and commercial users, worldwide, in numerous openly available implementations. The selection through a plethora of RISC-V processor cores and the mix of architectures and configurations combined with the proliferation of ML software frameworks for ML workloads, is not trivial. In order to facilitate this process, this paper presents a survey focused on the assessment of the ecosystem that entails RISC-V based hardware for creating a classification of system-on-chip (SoC) and CPU cores, along with an inclusive arrangement of the latest released frameworks that have supported open hardware integration for ML applications. Moreover, part of this work is devoted to the challenges that are concerned, such as power efficiency and reliability, when designing and building application with OSH in the AI/ML domain. This study presents a quantitative taxonomy of RISC-V SoC and reveals the opportunities in future research in machine learning with RISC-V open-source hardware architectures.

Keywords: RISC-V; open-source hardware; hardware accelerators; SoC; CPU; MCU; machine learning; deep learning; frameworks; survey

check for updates

Citation: Kalapothas, S.; Galetakis, M.; Flamis, G.; Plessas, F.; Kitsos, P. A Survey on RISC-V-Based Machine Learning Ecosystem. *Information* **2023**, *14*, 64. https://doi.org/10.3390/info14020064

Academic Editor: Valentina Casola

Received: 14 November 2022
Revised: 8 January 2023
Accepted: 18 January 2023
Published: 21 January 2023

1. Introduction

The emergence of the internet of things (IoT) has significantly contributed to the development and growth of edge computing and edge-AI systems. The plethora of embedded devices, sensors, and actuators have strengthened the developers' capabilities in designing cyber-physical systems in close proximity to the source of data. The edge computing paradigm also dictates the degree of exploitation of power efficient embedded systems. In more complex scenarios, where signal processing and machine learning algorithms are applied, an increased computation capacity, within a resource constrained hardware environment, is required.

Therefore, specialized hardware processors have been introduced to compensate the demand for efficient accelerators, in terms of power consumption and computational capacity, which is measured at the level of operations per second per watt, i.e., giga-operations per second per watt (GOPS/W). Typically, embedded central processing unit (CPU) cores are less optimized, compared to graphical processing units (GPUs) and thus, the former are used mainly in general data processing applications and the latter, in big-data analytics and convolutional neural network (CNN) training. In addition, field programmable gate arrays (FPGAs) and application specific integrated circuits (ASICs) have introduced significant improvements in power and computation efficiency [1,2]. However,

the ever-ongoing research and innovation towards more sophisticated and domain specific hardware accelerators have turned designers into exploring more massively open-source processor architectures, such as the ones based on the RISC-V instruction set architecture (ISA) standard.

Historically, RISC-based architectures have delivered more performance with less energy consumption and this feature provided an advantage compared to the complex instruction computer (CISC)-based ISA. In the early 1980s, acorn RISC machine (ARM) created a RISC-based processor, whereas Intel and AMD had already introduced the x86 processors, based on CISC ISA. More RISC-based commercial processors were also released, such as MIP [3], POWER [4] and SPARC [5]. In the 1990s, the x86 architecture became the mainstream architecture for desktop and server computers [6]. In recent times, however, ARM processors quickly dominated the mobile computer market and have also been used into high-performance computing (HPC) systems deployed in data-centers, where energy sustainability has always been a critical factor [7].

In parallel, further research in RISC architectures paved the way for the formulation of open-source RISC-based ISA, such as OpenRISC [8] and OpenSPARK [9], which are complete and fully open-source microprocessor designs, available in Verilog hardware description language (HDL) [10]. Eventually in 2010, a team at the UC Berkeley released the RISC-V ISA, a loyalty-free and open ISA, which was created with a vision to become the standard ISA for all computing devices [11]. The RISC-V ISA has a frozen base instruction set and can enable a set of optional extensions for domain specific application (DSA) SoCs to be created and thus, it is a more evolved and future-proof ISA. The demand for billions of cheap, energy efficient, and ubiquitous IoT devices has helped the proliferation of open-source hardware systems based on RISC-V cores and SoCs with ML capabilities. The RISC-V open standard is being publicly governed by the RISC-V Foundation. The Foundation is organized in open committees and technical working groups and has more than 1000 members from many industries and the academia [12].

Since 2010, the open RISC-V ISA has gained great popularity and nowadays, due to a critical mass of open-source hardware implementations, based on RISC-V, it is being demonstrated in a plethora of application domains [13–16]. Moreover, chip designs based on the RISC-V ISA, are of particular interest to the small and medium-sized enterprises (SMEs) and startups, due to non-existent license fees for proprietary intellectual property (IP) core blocks [17]. Further observations in accordance to Ref. [18] have highlighted the challenges for European Union's sovereignty in the context of open source hardware and, in particular, the market penetration rate of RISC-V in key sectors, such as IoT, industry, and automotive, to have reached 28%, 12%, and 10%, respectively.

In this paper, a survey on RISC-V based ML accelerators recently published in research journals, or commercially released, based on RISC-V, is presented. The survey is targeted at the various hardware implementations, in terms of available cores and SoCs, in conjunction with the software frameworks and software stacks for the SoC generation. The quantitative and qualitative results are discussed in the broader context of ML.

Furthermore, historical evolution of RISC processors and the overall development and paradigm shift towards open-source hardware, are presented. The RISC-V ISA base set and the custom extensions options, are also presented. Lastly, the various challenges and benefits of open-source hardware are discussed. To the best of our knowledge, this is the first study comparing RISC-V implementations in ML applications based on each of the surveyed software framework.

The rest of the paper is organized as follows. In Section 2, a background on the design challenges and benefits of RISC-V architecture, as well as an inclusive RISC-V base set and custom extension options, is presented. A comprehensive list of RISC-V CPU cores and SoCs, is presented in Section 3. In Section 4, a list of software frameworks with RISC-V SoC provisioning capabilities, is shown. In Section 5, case studies of RISC-V implementations in machine learning applications, are depicted. A further discussion and the final remarks are enclosed in Section 6.

2. RISC-V Background and Challenges

The need for low-power and low-cost accelerators has been significantly increased over the years. Specifically during the world-wide semiconductor supply-chain disruption, induced by the COVID-19 crisis and further deepened due to the Russia's invasion of Ukraine in early 2022, RISC-V-based ISA designs have helped open-source hardware solutions to emerge globally.

Equally, we have encountered challenges to keep up with the rapid increase of open-source hardware designs released in order to stay as inclusive as possible. The collection of specification and more in-depth information of commercially available CPU cores and SoCs was not easy either, as some of them were only available through a proprietary license. It is also worth stating that is a challenge to provide information on the tape-out process, therefore it will have to be considered only in future work.

2.1. Design Challenges

Primarily, there are noticeable benefits when embracing an open-source design, such as the one based on OSH. However, there is a considerable number of challenges for the designers to consider. Below a list of the identified benefits and challenges that are in particular related to RISC-V ISA, is presented.

- Licensing: The OSH definition is curated by the open-source hardware association (OSHWA), which extends not only to electronics, but also to machines and to other physical objects. The definition enables, although without imposing, new designs based on OSH to be released into the public domain where anyone can reuse, modify, and redistribute the designs. The CERN Open Hardware Licence version 2 consists of three variants (strongly-reciprocal, weakly-reciprocal, and permissive) [19] aligning with open-source software licensing schemes. In general, licensing has been providing permissions to individuals or commercial entities to reuse a design and potentially make a new product out of it. OSHWA offers a certification program for products commercially released under a OSH license. More in depth, the taxonomy of the licenses entail the free and permissive open-source licenses from the academia, such as the MIT, BSD and Apache 2.0 licenses, as well as proprietary, or non-permissive, licenses usually bound with specific IP hardware blocks, or entire designs;
- Power consumption: The importance of power efficiency has become dominant for hardware ML accelerator designs targeted towards edge computing applications. Typically, a higher operating frequency results in higher power consumption. Another critical factor is the complexity of the design, as the power estimation for RISC-V cores have shown variations depending the core architecture, operating frequency, and voltage [13]. Hence, design optimization is imperative to achieve a notably higher power efficiency;
- Security: Hardware and physical access security is of paramount importance in modern system architectures. RISC-V, as a relatively new ISA, has not yet gained research maturity in all the related security topics, such as memory protection, encryption verification, side-channel attack prevention, and control flow integrity [15]. However, RISC-V has settled a broad range of security features, which are profound in the hardware architecture, as well as many custom security extensions have been introduced to the RISC-V ISA. Further research in the aforementioned security topics and similar design considerations are, therefore, required;
- Interoperability: RISC-V, as a new ISA, has been far from establishing a universal adoption in terms of software development and operating system (OS) support. Specific RISC-V-based SoC boards are supported by some Linux flavors and real-time OS (RTOS) [20,21] and software developers must re-build existing code or start from scratch. The existence of the RISC-V GNU compiler toolchain [22] have supported developers in building new, or re-compile existing, C/C++ programs, as well as enable OS kernel support and device drivers for RISC-V hardware. Hence, this process has

been non-automatic and strenuous. Such an example, is the porting of the Android open-source project (AOSP) repositories to incorporate RISC-V hardware support [23].

2.2. RISC-V ISA Set of Extensions

The RISC-V ISA consists mainly of a base integer ISA that is carefully reduced to a minimal set of instructions, which is sufficient to provide a well defined target for compilers, assemblers, linkers, and operating systems. The ISA is modular and provides a lot of flexibility in keeping or removing the optional set of extensions, as well as in supporting the addition of custom extensions. This approach endorses designers to implement more efficient architectures without the need to include unused instruction set extensions.

The primary RISC-V ISA element is the base integer ISA (RV32I), which provides a 32-bit user-level address space and, as the name implies, it is used as a base in every implementation. There are also the RV64I and RV128I variants, which correspond to the register bit-width of 64-bit and 128-bit, respectively. Furthermore, there are also other non-overlapping standard extensions, as well as, other non-standard extensions that could be highly specialized, or could even conflict with other standard or non-standard extensions. For example, the RISC-V ISA has a little-endian memory system, but other non-standard variants can provide a big-endian or a bi-endian memory system.

In Table 1, a complete enumeration of standard extensions' set and a non-exhaustive set of custom extensions, is shown. The next most popular extension set is the 'M', which includes multiply and divide operations and is suitable for operations in machine learning applications, such as the ones included in network layers and activation functions within neural-networks. Data movement instructions are supported in 'A', the atomic extension. The 'F' and 'D' are the single/double-precision floating-point extensions. The set of standard extensions with the base integer is denoted with the letters IMAFD, or alternatively, with letter 'G' (RV32G). An example of a non-standard instruction set extension is 'V'; it supports a configurable vector unit to trade off the number of architectural vector registers and supported element widths against a fixed maximum available vector size. The 'V' extension can increase the vector size and dimension for the matrix multiplications, which has a distinct performance increase in convolutional neural networks. Similarly, non-standard instruction set extensions 'B' and 'P' have been introduced to enable bit manipulation and digital-signal processing (DSP) operations, respectively.

Table 1. RISC-V ISA standard and non-standard extensions.

Character	Name	Description
I	RV32I	Base integer instruction set
E	RV32E	Base integer for embedded (16-bit registers)
M	RV32IM	Multiply/Divide extension
A	RV32IMA	Atomic instructions
F	RV32IMAF	Single-precision floating-point
D	RV32IMAFD	Double-precision floating-point
G	RV32G	Shorthand for the IMAFD
Q	RV32Q	Quad-precision floating-point
C	RV32C	Compressed instructions
K	RV32K	Scalar cryptography
H	RV32H	Hypervisor extension
V	RV32V	Vector operations [1]
B	RV32B	Bit manipulation operations [1]
P	RV32P	DSP and packed SIMD instructions [1]

[1] Not ratified yet.

3. RISC-V Implementations

In this work, an extensive enumeration of the RISC-V cores and SoCs provided by the research community, or produced commercially, is presented. The process has been formulated around a quantitative analysis in order structure as a list of the supported hardware

specifications and ISA extensions, and also includes basic information and characteristics, such as year of debut, power consumption, operation frequency, performance, license, and implementation type (FPGA/ASIC). The architecture type, in terms of bit width, is also depicted. Furthermore, the type of HDL framework that a specific core, or SoC, has been design with and the license, under which the design is published, where applicable, has been provided.

All of the RISC-V designs that were considered in this work were released in 2018 and onwards. In addition, it is not a prerequisite for a design to have been taped-out in order to be listed. Therefore, RISC-V designs could have also been implemented either on FPGA as a synthesized soft processor or an ASIC build. Nonetheless, designs that have been realized only in simulation have not been considered in this study.

3.1. CPU Cores and SoCs

In Table 2, a list of RISC-V based processor implementations is presented.

In further detail, VexRiscv [24,25] is a very parameterizable RISC-V processor core implementation with support from the RV32I base instruction set that optionally can be extended up to the RV32IMAFDC instruction set. It is particularly optimized for FPGA by exempting from the use of vendor specific IPs and with a low-LUT footprint for enhanced portability. The bitstream for the core design is generated with the SpinalHDL [26] framework, exported into VHDL or Verilog. It can also be parameterized to be 32-bit or 64-bit FPU enabled. The framework can also provision two VexRiscv-based SoCs: the Briey SoC and Marex SoC. Both SoCs will generate more complex designs with SDRAM and VGA functionality, for example, which are capable of running Linux OS. Lastly, the VexRiscv architecture is very modular with optional plugins support for added functionalities, such as AES encryption/decryption acceleration, JTAG debug feature, and many others.

Rocket Core [27] is a scalar RISC-V core developed at the University of California, Berkeley (UCB), which implements the RV32G and RV64G ISA extensions. Optional FPUs and cache memory controllers with configurable sizes, are also supported. Rocket cores are emitted using the Rocket-chip generator, which is based in Scala and use the Chisel [28] compiler. Rocket custom co-processor (RoCC), is a custom vector architecture co-processor that is also supported by the framework as a non-standard RISC-V extension. Multiple instances of Rocket cores, cache controllers, and accelerators can be organized together in coherent-tiles to compose an assortment of SoC designs, using the Rocket-chip generator.

Berkeley Out-of-Order Machine (BOOM) [29] is another open-source processor derived from UCB. The main purpose was to build an educational tool for teaching undergraduate computer engineering classes. The initial architecture is based on Rocket and newer versions were developed based on the lessons learned from previous designs. The latest implementation is SonicBOOM (BOOM v3) [30], which has tackled bottlenecks from previous iterations as they were identified in the instruction fetch unit, execution backend, and load/store unit. SonicBOOM has been modeled on FPGA and it was also commercially fabricated at 1 GHz and achieved commercially-grade performance.

PicoRV32 [31] supports up to a RV32IMC instruction set and has a small footprint (<2 K LUT). This RISC-V design can also implement the RV32E core for even smaller and embedded (MCU) designs. The M extension is optionally supported via the Pico Co-Processor (PCPI) to feature non-branching instructions via a separate coprocessor implementation. PicoRV32 is highly size-optimized for FPGA as well as ASIC realizations.

Table 2. RISC-V core implementations by the academic community.

Core	Year	ISA	Freq. MHz	Perf. DMIPS/MHz	HDL	License	Type
VexRiscv	2018	RV32I[M][A][F][D][C]	200	1.38	SpinalHDL	MIT	FPGA
Rocket Core	2016	RV64I[M][A][F][D]	1000	1.72	Chisel	BSD	ASIC
SonicBOOM	2020	RV64GC	1000	3.93	Chisel3	BSD	ASIC
PicoRV32	2020	RV32I/E[M][C]	500	0.516	Verilog	ISC	FPGA
NEORV32	2020	RV32I/E[B][C][M][U][X]	150	0.952 [1]	VHDL	BSD	FPGA
NaxRiscV	2021	RV64I[M][A][F][D][C][S][U]	137	2.97	SpinalHDL	MIT	FPGA
NOEL-V	2020	RV64I[M][A][F][D][C][H][B]	100	4.69 [1]	VHDL	GPL	FPGA
ORCA	2016	RV32IM	122	0.98	VHDL	BSD	FPGA
SERV	2020	RV32I[M]	135	0.718	VHDL	ISC	FPGA
VROOM	2021	RV64IMAFDCHB[V]	-	6.5	Verilog	GPL3	FPGA
Ibex	2019	RV32I/E[M][C][B]	50	3.13 [1]	Verilog	Apache 2.0	FPGA

[1] in CoreMark/MHz. [] optionally enabled ISA.

NEORV32 [32] is a highly configurable and full-featured ISA extension core with complete documentation and multiple FPGA implementation examples to help beginners and more experienced users. The NEORV32 core can be integrated with various peripherals (UART, SPI, 1-Wire, PWM, watchdog) and a communication bus (Wishbone/AXI) in order to build a customizable SoC. The project is targeting microcontroller applications and is compatible with Zephyr [33] and FreeRTOS [34].

NaxRiscv [35] is an open-source core that supports Out-of-Order execution and is very scalar with many ISA extensions including 32-bit (RV32I) and 64-bit (RV64I) base integer instructions. The core can be built with SpinalHDL and has Linux and FreeRTOS support. Hardware implementations are currently integrated in Litex [36] and can run on FPGA. Low area usage and high operating frequency can be targeted with this RISC-V core. In a high performance configuration of a RV64 NaxRiscv running at 137 MHz on an Xilinx Artix 7 FPGA, 2.97 DMIPS has been achieved.

NOEL-V [37] is a RISC-V based core available with open-source license by a commercial company. It is designed to support both RV32 and RV64 architecture for embedded applications. The NOEL-V processor core and peripherals are included in a IP block library (GRLIB), which can be used to create a SoC. A list of pre-set configurations as well as custom configurations are available through the GRLIB. However, some of the configurations are available only under a commercial license, such as the GRFPU to enable the high-performance FPU, or the FT-FPGA, which includes fault-tolerance features for either FPGA or ASIC implementations. In a multi-core configuration for high-performance accelerator under a research project [38], its performance is reportedly one of the highest in this work.

ORCA [39] is a portable and FPGA-optimized RISC-V implementation with proprietary Vector extension support. The project is available as open-source and is currently maintained by the community, as the original author, company VectorBlox, has recently been acquired by Microchip. ORCA implementations have achieved a significant performance versus the area ratio.

One of the smallest RISC-V implementations was presented by the SERV [40] project. SERV is an open-source bit-serial CPU core and has been supported by Litex for a wide variety of FPGA boards. SERV is compatible with Zephyr RTOS and Linux. Moreover, SERV core can optionally add a multiplication and division unit (MDU), which complies with the M ISA extension.

VROOM [41] is a relatively new and still under development RISC-V implementation. VROOM is advertised as a high-end core that features the RV64IMAFDCHB(V) ISA extensions and offers a 2-way simultaneous multithreading (SMT). VROOM is available under the GNU public license v3 (GPL3) and has been tested with Linux running on AWS-FPGA instances.

Ibex [42] is a microcontroller-class RISC-V CPU core that has support for the RV32IEMC ISA extensions. Ibex has been previously developed under the name Zero-riscy [14], as part of the PULP platform cores [43], and currently has been contributed to lowRISC [44]. It is a production-grade CPU core with multiple FPGA implementations, as well tape-outs.

3.2. Commercial Cores

In Table 3, a list of commercially available systems with RISC-V cores has been provided. More information about the referenced cores is available below.

Table 3. RISC-V cores that are commercially available.

Core	Year	ISA	Freq. MHz	Perf. DMIPS/MHz	HDL	License	Type
SiFive E31	2019	RV32IMAC	320	1.61	Verilog	eval	FPGA
SiFive E51	2019	RV64IMAC	667	1.714	Verilog	eval	FPGA
XuanTie C910	2019	RV64IMAFDC	2500	6	Verilog [1]	-	ASIC
GD32VF103	2019	RV32IMAC	108	1.53	-	commercial	ASIC
K210	2018	RV64IMAFDC	400	0.8 [2]	-	commercial	ASIC
ESP32-S2	2019	RV32IMC	8	2.5 [3]	-	commercial	ASIC

[1] available via OpenC910. [2] combined performance in TOPS. [3] combined performance.

SiFive offers E31 [45], a RISC-V processor core that is distributed under commercial license for chip tapeout. An evaluation license is also available at no-cost for FPGA deployment, however, the RTL source code is obfuscated. E31 core has RV32IMAC ISA extension support with up to 8 cores coherency in a single design. It is claimed to be more efficient than ARM Cortex-M4 and it is considered suitable for a range of applications in edge computing and IoT.

The SiFive E51 [46] is an embedded RISC-V core which, for instance, is being used as a monitoring subsystem inside the PolarFire FPGA SoC [47]. A 64-bit architecture with RV64IMAC ISA extension is supported and can be integrated with up to 8 cores in full coherency.

XuanTie C910 [48] is a high-performance RISC-V CPU core that supports up to 4 cores at maximum 2.5 GHz operation frequency, and achieves 6 DMIPS/MHz. The RV64IMAFDC ISA extension is supported with a superscalar out-of-order pipeline. C910 is developed by T-Head, which is part of Alibaba, it has support for the ISA vector extension, while a set of non-standard instructions, for various task acceleration, is featured. C910 is supported by various Linux flavor ported to RISC-V architecture, such as Debian and Fedora ports. Recently, the OpenC910 [49], an open-source release of the C910, has been published under permissive Apache 2.0 license.

GD32VF103 [50] is a 32-bit RISC-V MCU SoC with RV32IMAC ISA extension support and a maximum operating frequency of 108 MHz. It is based on a single core architecture named Bumblebee core [51], which is jointly developed by Nuclei incorporating the N205 core [52]. A set of 12-bit capable ADC inputs and DAC outputs and other I/Os, such as UART and SPIs, are included in the design.

Part of the commercially available chips examined are a compilation of RISC-V and proprietary CPU, coprocessor, and accelerator cores. Therefore, the performance metrics collected cannot be fully assigned to RISC-V cores and they were not considered further in the evaluation. More notable examples are the ESP32-S2 [53] and K210 [54]. A RISC-V based co-processor supporting the RV32IMAC ISA extension together with a proprietary main processor IP are embedded in the ESP32-S2. K210 has a dual core 64-bit RISC-V MCU core, which complies with RV64IMAFDC ISA extension and has also a neural network processor (NPU) with a 0.8 TOPS claimed performance. In Figure 1, snapshots from a face detection inference demo running on K210, using a pre-trained AI model [55], is shown.

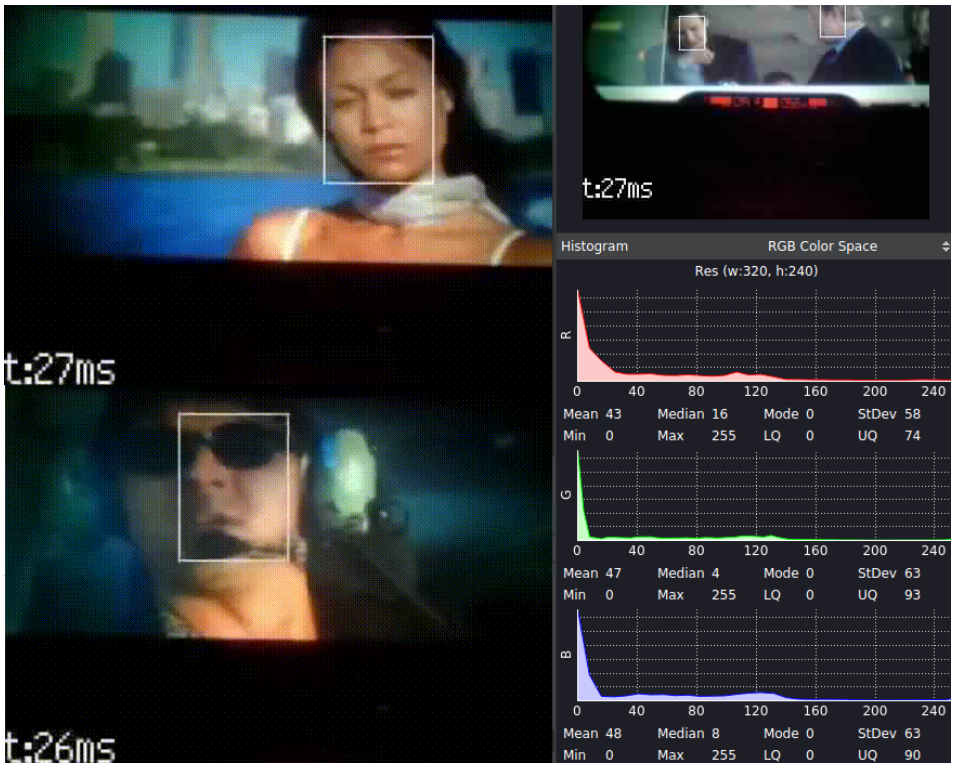

Figure 1. K210 face detection inference demo.

3.3. Resource Utilization

A summary of the resource utilization as publicized in the designers' documentation and benchmarks is discussed in this section. An aggregated list of the fundamental building blocks, such as look-up table, (LUT), flip-flop (FF), and block-RAM (BRAM), which are preoccupied per RISC-V core implementation on an FPGA, is presented in Table 4.

Table 4. RISC-V core implementations on FPGA.

Core	LUTs	FFs	BRAM	Notes
VexRiscv	504	505	0	RV32I, small config, Artix-7 [25]
Rocket Core	4413	2205	5.5	Rocket core, Zynq UltraScale + MPSoC [56]
SonicBOOM	148,500	-	-	2-wide superscalar Medium BOOM, 1 core [57]
PicoRV32	917	583	0	regular config, Arty A7-100T [31]
NEORV32	1328	678	0	rv32i_Zicsr, Cyclone IV [32]
NaxRiscV	11,600	9200	11.5	RV32IMASU, Arty A7-100T [35]
NOEL-V	22,960	10,350	28	RV64IMA, Arty A7-100 [37]
ORCA	1350	746	1	RV32I, XC7Z020 [58]
SERV	436	375	0	RV32I, Arty A7-100T [40]
VROOM	-	-	-	not provided
Ibex	600	1000	48	RV32I, Arty A7-100T [42]
SiFive E31	3614	1642	0	RV32IMAC, XC7Z020 [58]
OpenC910	669,902	235,730	347	RV64GC, Virtex UltraScale [59]

In Figure 2, a data correlation between the hardware synthesis LUTs utilization and the performance, in terms of DMIPS/MHz, for the RISC-V cores, is visually illustrated. Typically, for high performance cores, high resource utilization is also inferred. VROOM core achieves the best performance, but no design details have been provided to highlight the use of hardware resources required to reach it. From the results, however, it is also formulated that some implementations can achieve higher performance and LUTs utilization ratio. This performance efficiency metric has been evaluated for all the RISC-V

implementations. The most efficient RISC-V implementation in our study has been the ibex core, followed by VexRiscv and SERV; ibex has the second highest BRAM utilization, however.

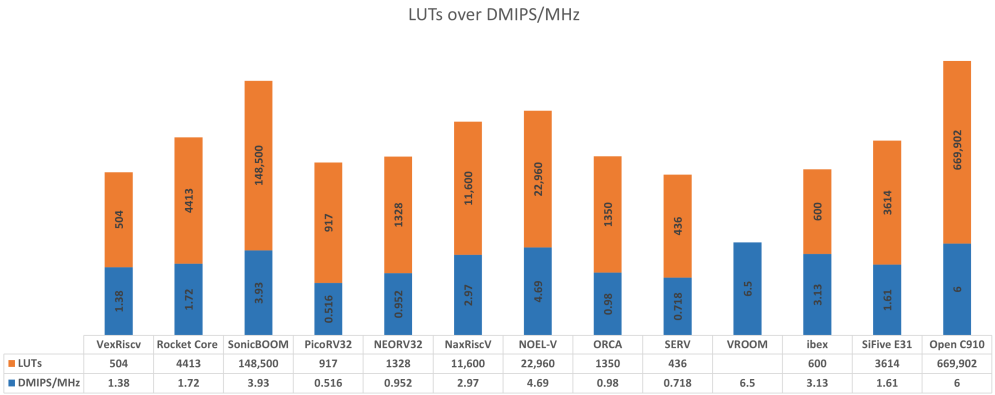

Figure 2. Hardware resource utilization and performance comparison.

4. Software Frameworks and Stacks

This section is dedicated to the frameworks and tools for designing hardware accelerators in ML applications that incorporate the RISC-V architecture. The criteria for accepting a specific framework was based on the year of its release; no frameworks released earlier than 2018 were added in this survey. In the following, we present the examined frameworks.

4.1. TVM

Tensor Virtual Machines (TVM) [60,61] is an open-source ML framework created by Apache with the goal to help engineers optimize and run ML models effectively on a hardware-agnostic way. A quote from the TVM developers' community, states:

> "The vision of the Apache TVM Project is to host a diverse community of experts and practitioners in machine learning, compilers, and systems architecture to build an accessible, extensible, and automated open-source framework that optimizes current and emerging machine learning models for any hardware platform."

TVM offers the infrastructure layer needed to automatically develop and improve ML models on a plethora of hardware backend (CPUs, GPUs, MCUs, FPGAs) and compile them into minimum deployable modules to attain better performance. The optimization is achieved via either the AutoTVM or the AutoScheduler auto-tuning modules that search for the best execution schedule, of the ML model graph, compared to cost models and on-device measurements.

The Versatile Tensor Accelerator (VTA) [62], is an ML hardware accelerator, provided as an extension of the TVM framework, which aims to advance deep learning and hardware innovation. It is an open, generic, and customizable RISC-like programmable accelerator inspired by mainstream deep learning accelerators. Together, TVM and VTA form an end-to-end accelerator-centric deep learning framework for researchers and practitioners to incorporate their optimizations and co-design techniques.

Despite TVM having demonstrated good performance across a wide range of models on traditional operating systems, microTVM [63] is using the TVM flow to run models on bare-metal devices. It depends only on the C standard library, and does not require an operating system, virtual memory, or advanced programming language to execute. It follows two execution models, the Host-Driven and the Standalone. In the first model, a ML model graph executor is created on a host that drives the device, which executes the computations needed through an remote procedure call (RPC) link. Whereas in the Standalone execution, the graph executor is instantiated and executed directly on the device.

Currently, microTVM is tested against Cortex-M microcontrollers with the Zephyr RTOS on various development boards and against QEMU emulator, but the developers claim that it is flexible and portable to other processors, such as RISC-V, without the prerequisite of Zephyr RTOS.

4.2. CFU Playground

Custom function unit (CFU) Playground [64,65] is a full-stack open-source framework that aims to enable rapid prototype development of ML accelerators for FPGA systems. It was initially developed by a team at Google, but it is not an officially supported Google project. The framework addresses individuals (engineers, researchers, students) who need a tool to design and evaluate hardware accelerators, as an FPGA-based soft processor, to increase the performance of ML model computation. The goal is to abstract most of the underlying infrastructure complexity and provide users with a simplistic way to develop hardware enchantments for ML tasks. The hardware accelerators are implemented as a CFU for the soft processor, a RISC-V CPU. CFU is a digital logic circuit tightly coupled with the CPU that adds a custom instruction ISA using a standardized format defined by the RISC-V FPGA Soft Processor Working Group. The details of that format are out of the scope of this work, but the brief aim of these instructions is to accelerate the tasks widely used in ML applications, formerly executed in software. These tasks can be extended from simple mathematical operations, such as multiply and accumulate (MAC), to more complex operations, such as a complete convolution function.

The CFU Playground consists of a set of configuration options on three major components, namely software, gateware, and hardware. A short description of these configurations is presented in the paragraphs below.

The gateware is built on the Litex framework, which provides the infrastructure to create a System-on-Chip (SoC) on FPGA, and it is comprised of a soft-core CPU and various peripherals. The soft-core used in CFU Playground is based on VexRiscv [24] with a CFU extension, making it a highly configurable CPU core, which makes it easy to add or remove different features for performance and functionality such as pipeline, caches, etc. Despite the fact that VexRiscv is currently the only soft-core supported, in theory any CPU supported by the Litex framework could alternatively be used. More in detail, the CFU is a small custom logic circuit designed to extend the soft-core and accelerate discrete operations. The communications between the CPU and CFU follows the RISC-V R-format interface, a command–result protocol. The CPU uses the custom functions' op code allocated to the CFU by the user. When the CPU executes the specific op code, it passes the functions arguments, i.e., the contents of two registers to the CFU waits for a response and puts the result back into another register. A notable architecture decision is that CFU has not introduced direct access to memory, but is rather dependent on CPU to move data. CFU may be described in Verilog or any other tool that is capable to output Verilog, such as, amaranth and chisel. Ultimately, the framework automatically uses a set of tools to synthesize, place, and route the complete design into an FPGA. These tools are mainly open-source, but a commercial toolchain, supplied by the FPGA manufacturer, can also be used.

The SoC produced by the gateware following the aforementioned process can be implemented in various FPGA hardware platforms. The minimum requirements for the board and its FPGA device are the following: (a) there must be the capability to create a communication link such as TTY/UART to interact with the program on the board, (b) there must be sufficient resources on the FPGA device to implement the VexRiscv CPU with the CFU, and (c) the system must have enough RAM and ROM memory to hold code and data of the ML model. More specifically, the CFU Playground currently supports the Xilinx 7-Series FPGAs, as well as the Lattice iCE40, ECP5, and CrossLink FPGAs and it has been tested on the Arty A7-35T/100T, iCEBreaker, Fomu, OrangeCrab, ULX3S, and Nexys video boards. As previously discussed, in theory any board supported by the Litex

framework can be used in CFU Playground, by creating a hardware description in Litex of a non-disclosed (proprietary) prototype board, as an example.

Alternatively, if none of the aforementioned hardware platforms are at disposal, Renode [66], an open-source simulation framework developed by Antmicro, can be used to simulate the physical hardware system. A single or multiple CFU verilated model(s) can be used in co-simulation with renode to test the end-to-end flow of the embedded software, on desktop environment.

The CFU Playground software is targeting VexRiscv CPU, which is a standard 32 bit RISC-V CPU and the open-source GCC C/C++ toolchain together with picolibc is used to compile the executables. The framework includes, (a) Tensorflow Lite for microcontrollers for ML model inferencing, (b) example models and test data to experiment, (c) tools for profiling and benchmarking, and (d) customization examples of tensorflow kernels, and many others.

As a conclusion, CFU Playground is a full-stack framework that provides to developers, out-of-the-box, an open-source design flow to explore the design space between the CPU and a tightly-coupled CFU to improve ML model execution in terms of speed, space, and power-energy efficiency.

4.3. Chipyard

The Chipyard framework [67], developed at the University of California, Berkeley, is a unified SoC design, simulation, and implementation environment. It comprises a series of independently developed, high configurable, open-source IP blocks, which can easily be combined to form a SoC. The framework provides flows to verify and validate the design both in FPGA hardware and software simulation, while it concludes a workload management system that produces workloads for the design to exercise the system SoC. Ultimately, the tool can drive the design through a very-large scale integration (VLSI) design flow to produce tapeout data for various technologies. The Chipyard framework is, in simple words, a collection of tools to address the main process flows in SoC design, including, (a) the front-end RTL design, (b) system validation-verification, and (c) back-end register-transfer level (RTL) chip physical design.

The front-end is built around the Rocket Chip SoC generator [27], which is a chisel-based parameterized hardware generator library that enables the generation of heterogeneous SoC based on different configurations. The IP blocks written in other hardware description languages, such as Verilog, can be included in the design via chisel wrappers. This library is enhanced with a large number of open-source IP generators, which facilitates the construction of complete digital SoC systems. Indicative examples of those IPs are, the Berkeley Out-of-Order Machine (BOOM) [29], the Ariane core [68], the Hwacha Vector-Fetch Architecture [69], dsp modules, domain-specific accelerators, memories modules, and peripherals. Chipyard accomplishes to support multiple concurrent design flows from the same RTL code base by using Flexible-Internal-Representation for RTL (FIRRTL) [70], a custom intermediate representation transformation to drive each flow used at different stages of the design cycle.

In regards to software-based RTL simulation, Verilator is an open-source and widely used simulator in the industry, however, other proprietary commercial simulators are also supported by the framework. System level simulation is achieved by utilizing makefile wrappers to generate executables based on SoC configuration, which simulate the core design with emulated peripherals. In most cases, a host is used to bring up the simulated SoC and load programs, but configurations that can boot the SoC standalone, through a boot ROM, are also supported.

In addition to software RTL simulation, Chipyard provides a flow for FPGA-accelerated simulation. It is based on FireSim [71], which is an open-source simulation platform based on AWS EC2 public cloud for system level validation and evaluation. The above method differs from native FPGA prototyping, which depends on device peripherals performance

and provides a deterministic evaluation system environment with accurate models timing behavior for both the core, the peripherals, and the I/Os of the SoC.

Moreover, the Chipyard framework provides the FireMarshal [72] software workload generation tool to enable software development at all design stages. Software engineers can initiate development as soon as the functional model is available using tools such as the Spike RISC-V ISA simulator [73] and QEMU emulator. A versioned set of standard tools (e.g., GNU toolchain, Spike, QEMU), as well as a set of non-standard tools for custom extension, or IP blocks, are provided with FireMarshal. Lastly, several examples and templates of Linux based workloads to depict the speed-up of software development, are included with the specific tool.

4.4. Open ESP

ESP [74] was developed by the system-level design (SLD) group at Columbia University and is, according to the developers:

> "An open-source research platform for heterogeneous system-on-chip design that combines a scalable tile-based architecture and a flexible SLD methodology."

ESP makes the software and hardware integration feasible by providing multiple design flows for accelerator development. The platform combines several open-source hardware components, such as RISC-V or Sparc processor cores with a set of open-source and commercial CAD tools.

Currently, multiple design flows that allow for the creation of a rich library of heterogeneous hardware components is already supported by ESP methodology. As a first option, hardware designers can develop components with a traditional RTL design flow (VHDL, Verilog, chisel, etc.). As a second option, designers can leverage the ESP automation tools, in combination with commercial high-level synthesis (HLS) tools, such as Xilinx Vivado HLS or Catapult HLS, to create accelerators in C-like languages. At the ML application domain, the open-source hls4ml [75] tool is integrated in the platform, which supports application developers to generate accelerators from models developed in frameworks, such as Keras or Pytorch. This list of alternative accelerator design flows, which grows continuously, allow designers to choose the abstraction level and specification language to suite their skills and technical background. The tool also enables the integration with third-party IP blocks through the AXI protocol, while the platform includes examples for the Ariane RISC-V processor core [68] and the NVIDIA deep learning accelerator (NVDLA) [76] to validate the specific feature. The platform offers a graphical user interface (GUI) that guides the designers through an interactive floor planning of the SoC and offers the capability for rapid prototyping of the design on various FPGAs platforms. The ESP architecture is tiled-based and the structure of the SoC is determined by the number and the mix of tiles. A tile may contain a processor core, an accelerator, memory, or scratchpad and I/O, and all the tiles are connected with multi-plane network-on-chip (NoC).

ESP also provides an application programming interface (API) library to simplify the ML application development. The API exposes three functions to the programmer to transparently invoke the accelerators with automatically provisioned Linux device drivers. In practice, the software execution of a computational intensive kernel can be accelerated in hardware, when replacing it with a single API function call. More specifically, esp_run() defines the configuration arguments to determine which accelerator(s) to invoke. The rest of the API functions (i.e., esp_alloc() and esp_free()) are responsible for the data memory allocation and exchange between the processor(s) and accelerator(s) upon an efficient zero-copy data policy memory access in order to not compromise the software performance. The combination of the ESP software stack and automatic generation of device drivers for accelerators, makes the application hardware acceleration as transparent as possible to software designers.

4.5. NVDLA

The NVDLA project is an open-source hardware and software platform that provides a standardized way to accelerate deep learning network inference. It is released under the NVIDIA open license. It is designed to be modular, scalable, and highly configurable, offering a wide range of performance levels from small cost-effective to larger performance-oriented devices. The hardware project consists of a Verilog based RTL synthesis and simulation and a transaction-level modeling (TLM) System C model, which is used in software development, system integration, verification, and testing. The software part includes an on-device software stack and the framework for training new models, or convert existing ones, to be executed with the on-device software. The hardware architecture is based on the principal that implies that the majority of compute effort on deep learning inference is based on mathematical operations such as convolution, activation, pooling, and normalization functions. Thus, these functions are accelerated in hardware as separate components, while a memory and data reshape unit is also included for tensor reshape and memory-to-memory copy operations' acceleration. These blocks are separate and independently configurable, and therefore the designer can flexibly tune the accelerator for the specific needs of the application. Every component in the NVDLA architecture has a role to support a specific inference operation of a neural network; for example, the convolution core maps onto TensorFlow operations, such as tf.nn.conv2d. In documentation, TensorFlow operations were provided as examples of components mappings, nonetheless, the NVDLA supports additional deep learning frameworks as well.

The accelerator design uses standard protocols to interface with the rest of the system, i.e., (a) a control channel with a register file and an interrupt interface and (b) a pair of AXI bus interfaces to connect with memory. The first memory interface connects to the system main memory and I/O peripherals, including DRAM. The second memory interface is optional and allows the connection to high-bandwidth memory dedicated to NVDLA.

NVDLA implementations generally follow two models: (a) the headless or "small", where unit-by-unit hardware management happens on a main processor and (b) the headed, or "large", which delegates tasks to a companion, tightly coupled, microcontroller.

The "small" model is a good fit for cost-sensitive devices, where cost, area, and power are the primary concerns. Savings are assured through scaling down NVDLA implementation and sacrificing system performance. Neural network models are fine tuned to consume less storage and take less time to load and process, which in turn enables a scaled down NVDLA implementation when system performance is not the main issue.

The second, or "large", model emphasizes high-performance and versatility. In such a system, the NVDLA is capable of performing inference on different network topologies and deserialize execution, i.e., execute multiple tasks at once. These requirements make the use of a second memory interface, a necessity for high-bandwidth SRAM and dedicated control coprocessor connection, in order to offload the main processor. The additional SRAM serves as a cache for NVDLA and may be shared with other high-performance components, such as computer vision hardware, to reduce main memory traffic. Many general-purpose processors can serve as system coprocessor, i.e., RISC-V, or ARM-Cortex, since it will be mainly responsible for scheduling and fine-grained programming of the NVDLA hardware.

NVDLA encompasses a complete software ecosystem to support the aforementioned operations, including the on-device software stack and training support facilities, to build new, or convert existing, models into a form that is compatible with NVDLA software.

The software subsystem is comprised of (a) the compilation tools for model generation and (b) the runtime environment for loading and executing neural networks on NVDLA. The compilation tools include a compiler, responsible for generating a sequence of hardware layers optimized for a specific NVDLA configuration, and a parser, which creates an intermediate representation from a pre-trained neural network to be passed to the compiler. The runtime environment is responsible for model execution on the compatible NVDLA hardware. It can be divided into two components (a) the User Mode Driver (UMD), which

loads the output produced by the compiler, also called "loadable", and submits inference jobs, and (b) the Kernel Mode Driver (KMD), which handles the scheduling of layers operations on deep learning with autoencoders (DLNA) hardware and configures each correspondent function block.

Lastly, sample platforms are offered out-of-the-box and allow users to evaluate and test NVDLA and develop software prior of integration onto a larger, industrial grade SoC design. The first is a simulation platform based on GreenSocs qbox [77]. In this approach, a QEMU CPU model is combined with the NVDLA System C model and forms a register–level accurate system, on which software development and debugging is feasible, while a Linux kernel-mode driver and a user space test utility are available to run on this platform. The second platform maps the Verilog model onto an FPGA, which can serve as an example on instantiating NVDLA in a real design. The FPGA system is then deployed in an Amazon EC2 "F1" environment, which can be leased with a pay-by-hour rate, but since this is a Xilinx-based platform, porting to other Virtex-family devices should be possible.

5. Case Studies of RISC-V in Machine Learning

In this last section, a list of existing use cases that validate the effectiveness of open-source hardware accelerator implementations based on existing RISC-V core architecture and frameworks discussed in Section 4, as well as other custom RISC-V implementations in machine learning applications, is discussed. The aggregated list of the hardware accelerators' case studies described further in this section, is provided in Table 5 below.

Table 5. Case studies of RISC-V hardware accelerators in machine learning.

Case Study	Framework	Core
[64]	CFU	VexRiscv
[78]	Chipyard	BOOM
[79]	OpenESP	Ariane
[80]	NVDLA	Rocket
[81]	VHDL + LLVM/Clang	Arrow
[82]	RTL	RISC-V^2
[83]	VHDL	Arnold
[84]	GAP-8 SDK	GAP-8

In Ref. [64] the authors of the CFU playground have developed a framework that generates hardware accelerators based on VexRiscv core. In image classification tasks, running the MobileNetV2 neural network architecture, an up to 55× times speedup has been measured in 1 × 1 convolutions. In keyword spotting application, the CFU has achieved up to 75× times faster inference, in 1 × 1 convolutions as well.

The hardware accelerator included in a SoC design [78] based on Chipyard framework have exploited the RISC-V architecture with Out-of-Order execution capabilities and have demonstrated 282× times MOPS improvement, in relation to an in-order based RISC-V core, for DNN and vector workloads.

Open ESP was exploited in Ref. [79] in order to integrate the Ariane RISC-V core together with other third-party IP cores, such as multiple NVDLA instances for expedited DNN inference. All tiles were tightly coupled into an SoC architecture, which also leveraged a scalable memory hierarchy and network-on-chip, and were realized onto an FPGA.

In Ref. [80], the authors implemented LeNet-5 running on a RISC-V SoC exploiting the NVDLA framework, and measured decent acceleration performance at low power. An outstanding 4647× times increase in inference performance compared to K210, a 2-core RISC-V hardware board, was shown.

Arrow [81] is a RVV-based accelerator with energy and performance benefits for edge machine learning inference applications. Arrow is a configurable hardware accelerator with partial RISC-V vector ISA extension implementation. The authors have demonstrated up to

78× times faster performance and up to 99% less energy consumption when implemented on a specific Xilinx FPGA.

RISC-V^2 [82] is a scalable vector processor design with significant performance gains evaluated in thorough CNN benchmarks. In this research work, three distinct techniques have been proposed to increase performance: a register remapping scheme for dynamic register allocation; a decoupled execution scheme between execution and memory-access instructions; and a hardware support for vector reduction operations.

Arnold [83] is a RISC-V MCU based on the PULP/Riscy core augmented with an eFPGA that achieves great performace-to-power ratio compared to other MCUs. The authors have claimed 3.4× times better performance and 2.9× times better energy efficiency than other fabricated heterogeneous and reconfigurable SoCs of the same class, and an overall capability of 600 MOPS.

Lastly, GAP-8 [84] is an ultra-low power RISC-V platform that has evidently showcased energy efficiency improvements through its parallel architecture, in complex applications. The are 8 RISC-V cores included in the SoC, which have exhibited 10× times performance improvement and adequate energy efficiency at ultra-low power when parallelism has been fully exploited in vector computing.

6. Conclusions

In this paper, we presented an introduction to the RISC-V architecture and the ISA extensions included in the standard, or which are available as custom extensions. The list of RISC-V cores ranges from the academic community to the industry, and intends to be as inclusive as possible and to present not only legacy, but more recent architectures as well. As a result, a total number of 17 RISC-V cores were presented and analyzed. From the analysis of the cores' characteristics, interesting results, with definitive performance and resource utilization implications, have been showcased.

In addition, an extensive presentation of the available software frameworks that take advantage of the RISC-V architecture is of higher importance in this work; for example, the role of RISC-V to design and scale up AI/ML training workloads, or alternatively, to apply model quantization and pruning techniques and deploy these trained models onto RISC-V based edge systems and run inference. To the best of our knowledge, such an inclusive, although non-exhaustive survey has not yet been contributed to the literature.

Conclusively, the goal of the analysis is to put the RISC-V architecture into the perspective of hardware designers and developers and to help them acquire a better grasp of the most prominent implementations in combination with the most known ML frameworks, which, all combined, form a large hardware and software ecosystem.

Author Contributions: Conceptualization, S.K. and M.G.; methodology, S.K.; software, S.K.; validation, S.K., M.G. and G.F.; formal analysis, S.K.; investigation, S.K. and M.G.; resources, S.K.; data curation, S.K.; writing—original draft preparation, S.K. and M.G.; writing—review and editing, P.K.; visualization, S.K.; supervision, P.K. and F.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

CPU	Central Processing Unit
MCU	Micro Controller Unit
UART	Universal Asynchronous Receiver-Transmitter
SPI	Serial Peripheral Interface
MOPS	Milion Operations Per Second
GOPS	Giga Operations Per Second
TOPS	Tera Operations Per Second
SDK	Software Development Kit

References

1. Kalapothas, S.; Flamis, G.; Kitsos, P. Efficient Edge-AI Application Deployment for FPGAs. *Information* **2022**, *13*, 279. [CrossRef]
2. Kalapothas, S.; Flamis, G.; Kitsos, P. Importing Custom DNN Models on FPGAs. In Proceedings of the 2021 10th Mediterranean Conference on Embedded Computing (MECO), Budva, Montenegro, 7–10 June 2021; pp. 1–4.
3. Kane, G.; Heinrich, J. *MIPS RISC Architectures*; Prentice-Hall, Inc.: Hoboken, NJ, USA, 1992.
4. Karkhanis, T.; Moreira, J.E. IBM Power Architecture. 2011. Available online: https://dominoweb.draco.res.ibm.com/reports/rc25146.pdf (accessed on 9 October 2022).
5. Agrawal, A.; Garner, R.B. SPARC: A scalable processor architecture. *Future Gener. Comput. Syst.* **1992**, *7*, 303–309. [CrossRef]
6. McAllister, N. Intel x86 CONQUERED THE WORD. (cover story). *InfoWorld* **2005**, *27*, 24–31.
7. Hussain, S.M.; Wahid, A.; Shah, M.A.; Akhunzada, A.; Khan, F.; Arshad, S.; Ali, I. Seven Pillars to Achieve Energy Efficiency in High-Performance Computing Data Centers. In *Recent Trends and Advances in Wireless and IoT-Enabled Networks*; EAI/Springer Innovations in Communication and Computing; Jan, M., Khan, F., Alam, M., Eds.; Springer: Cham, Switzerland, 2019; pp. 93–105. [CrossRef]
8. OpenRISC Project. Available online: https://openrisc.io/ (accessed on 7 October 2022).
9. OpenSPARC Overview. Available online: https://www.oracle.com/servers/technologies/opensparc-overview.html (accessed on 7 October 2022).
10. *IEEE Std 1800-2005*; IEEE Standard for SystemVerilog: Unified Hardware Design, Specification and Verification Language. IEEE: Piscataway, NJ, USA, 2005; pp. 1–648. [CrossRef]
11. Asanović, K.; Patterson, D.A. Instruction sets should be free: The case for risc-v. In *Technical Report UCB/EECS-2014-146*; EECS Department, University of California: Berkeley, CA, USA, 2014.
12. RISC-V Foundation. Available online: https://live-risc-v.pantheonsite.io/technical/technical-forums/ (accessed on 7 October 2022).
13. Dörflinger, A.; Albers, M.; Kleinbeck, B.; Guan, Y.; Michalik, H.; Klink, R.; Blochwitz, C.; Nechi, A.; Berekovic, M. A comparative survey of open-source application-class RISC-V processor implementations. In Proceedings of the 18th ACM International Conference on Computing Frontiers, Virtual Event, Italy, 11–13 May 2021; pp. 12–20. [CrossRef]
14. Schiavone, P.D.; Conti, F.; Rossi, D.; Gautschi, M.; Pullini, A.; Flamand, E.; Benini, L. Slow and steady wins the race? A comparison of ultra-low-power RISC-V cores for Internet-of-Things applications. In Proceedings of the 2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS), Thessaloniki, Greece, 25–27 September 2017; pp. 1–8. [CrossRef]
15. Lu, T. A survey on risc-v security: Hardware and architecture. *arXiv* **2021**, arXiv:2107.04175.
16. Di Mascio, S.; Menicucci, A.; Gill, E.; Furano, G.; Monteleone, C. Leveraging the Openness and Modularity of RISC-V in Space. *J. Aerosp. Inf. Syst.* **2019**, *16*, 454–472. [CrossRef]
17. Palmer, C. Simplified Instruction Set Architecture Accelerates Chip Development—And Wins the 2022 Draper Prize. *Engineering* **2022**, *17*, 7–9. [CrossRef]
18. Recommendations and Roadmap for European Sovereignty on Open Source Hardware, Software and RISC-V Technologies. Available online: https://digital-strategy.ec.europa.eu/en/library/recommendations-and-roadmap-european-sovereignty-open-source-hardware-software-and-risc-v (accessed on 9 October 2022).
19. CERN Open Hardware License Version 2. Available online: https://ohwr.org/project/cernohl/wikis/Documents/CERN-OHL-version-2 (accessed on 15 October 2022).
20. RISC-V Ubuntu Wiki. Available online: https://wiki.ubuntu.com/RISC-V (accessed on 15 October 2022).
21. Using FreeRTOS on RISC-V Microcontrollers. Available online: https://www.freertos.org/Using-FreeRTOS-on-RISC-V.html (accessed on 15 October 2022).
22. RISC-V GNU Compiler Toolchain. Available online: https://github.com/riscv-collab/riscv-gnu-toolchain (accessed on 15 October 2022).
23. RISC-V Android Port. Available online: https://github.com/riscv-android-src/riscv-android (accessed on 15 October 2022).
24. Charles Papon with VexRiscv Was Awarded $6,000 USD. Check out VexRiscv on GitHub. Available online: https://github.com/SpinalHDL/VexRiscvSoftcoreContest2018/ (accessed on 20 October 2022).
25. A FPGA Friendly 32 Bit RISC-V CPU Implementation. Available online: https://github.com/SpinalHDL/VexRiscv (accessed on 20 October 2022).
26. Scala Based HDL. Available online: https://github.com/SpinalHDL/SpinalHDL (accessed on 20 October 2022).

27. Asanovic, K.; Avizienis, R.; Bachrach, J.; Beamer, S.; Biancolin, D.; Celio, C.; Cook, H.; Dabbelt, D.; Hauser, J.; Izraelevitz, A.; et al. *The Rocket Chip Generator*; Technical Report UCB/EECS-2016-17; EECS Department, University of California: Berkeley, CA, USA, 2016; Volume 4.
28. Bachrach, J.; Vo, H.; Richards, B.; Lee, Y.; Waterman, A.; Avižienis, R.; Wawrzynek, J.; Asanović, K. Chisel: Constructing Hardware in a Scala Embedded Language. In Proceedings of the 49th Annual Design Automation Conference (DAC '12), San Francisco, CA, USA, 3–7 June 2012; pp. 1216–1225. [CrossRef]
29. Asanovic, K.; Patterson, D.A.; Celio, C. *The Berkeley Out-of-Order Machine (Boom): An Industry-Competitive, Synthesizable, Parameterized Risc-V Processor* ; Technical Report; University of California: Berkeley, CA, USA, 2015.
30. Zhao, J.; Korpan, B.; Gonzalez, A.; Asanovic, K. Sonicboom: The 3rd generation berkeley out-of-order machine. In Proceedings of the Fourth Workshop on Computer Architecture Research with RISC-V, Virtual Workshop, 29 May 2020 ; Volume 5.
31. PicoRV32—A Size-Optimized RISC-V CPU. Available online: https://github.com/YosysHQ/picorv32 (accessed on 20 October 2022).
32. The NEORV32 RISC-V Processor. Available online: https://github.com/stnolting/neorv32 (accessed on 20 October 2022).
33. The Zephyr Project. Available online: https://www.zephyrproject.org/ (accessed on 20 October 2022).
34. FreeRTOS—A Real-Time Operating System for Microcontrollers. Available online: https://www.freertos.org/index.html (accessed on 20 October 2022).
35. NaxRiscv—An Open-Source OoO Superscalar Softcore. Available online: https://github.com/SpinalHDL/NaxRiscv (accessed on 20 October 2022).
36. Kermarrec, F.; Bourdeauducq, S.; Badier, H.; Le Lann, J.C. LiteX: An open-source SoC builder and library based on Migen Python DSL. In Proceedings of the OSDA 2019, Colocated with DATE 2019 Design Automation and Test in Europe, Florence, Italy, 25–29 March 2019.
37. NOEL-V—A Synthesizable VHDL Model of a Processor That Implements the RISC-V Architecture. Available online: https://www.gaisler.com/index.php/products/processors/noel-v (accessed on 20 October 2022).
38. Wessman, N.J.; Malatesta, F.; Ribes, S.; Andersson, J.; García-Vilanova, A.; Masmano, M.; Nicolau, V.; Gomez, P.; Rhun, J.L.; Alcaide, S.; et al. De-RISC: A Complete RISC-V Based Space-Grade Platform. In Proceedings of the 2022 Design, Automation & Test in Europe Conference & Exhibition (DATE) (DATE '22), Antwerp, Belgium, 14–23 March 2022; pp. 802–807.
39. ORCA—RISC-V by VectorBlox. Available online: https://github.com/kammoh/ORCA-risc-v (accessed on 20 October 2022).
40. SERV—The SErial RISC-V CPU. Available online: https://github.com/olofk/serv (accessed on 20 October 2022).
41. VRoom! RISC-V CPU. Available online: https://github.com/MoonbaseOtago/vroom (accessed on 20 October 2022).
42. Ibex Is a Small 32 Bit RISC-V CPU Core, Previously Known as Zero-Riscy. Available online: https://github.com/lowRISC/ibex (accessed on 20 October 2022).
43. Rossi, D.; Conti, F.; Marongiu, A.; Pullini, A.; Loi, I.; Gautschi, M.; Tagliavini, G.; Capotondi, A.; Flatresse, P.; Benini, L. PULP: A parallel ultra low power platform for next generation IoT applications. In Proceedings of the 2015 IEEE Hot Chips 27 Symposium (HCS), Cupertino, CA, USA, 22–25 August 2015; pp. 1–39.
44. lowRISC—Open to the Core. Available online: https://lowrisc.org/ (accessed on 20 October 2022).
45. SiFive E31 Standard Core. Available online: https://www.sifive.com/cores/e31 (accessed on 20 October 2022).
46. SiFive E51 Standard Core. Available online: https://static.dev.sifive.com/E51-RVCoreIP.pdf (accessed on 20 October 2022).
47. PolarFire® SoC FPGA. Available online: https://www.microchip.com/en-us/products/fpgas-and-plds/system-on-chip-fpgas/polarfire-soc-fpgas (accessed on 20 October 2022).
48. Chen, C.; Xiang, X.; Liu, C.; Shang, Y.; Guo, R.; Liu, D.; Lu, Y.; Hao, Z.; Luo, J.; Chen, Z.; et al. 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 Proceedings of the 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA), Valencia, Spain, 30 May–3 June 2020; pp. 52–64.
49. OpenXuantie—OpenC910 Core. Available online: https://github.com/T-head-Semi/openc910 (accessed on 20 October 2022).
50. GD32VF103 Series—GD32 RISC-V Microcontrollers. Available online: https://www.gigadevice.com/products/microcontrollers/gd32/risc-v/mainstream-line/gd32vf103-series/ (accessed on 20 October 2022).
51. Bumblebee ProcessorCore ISA. Available online: https://www.rvmcu.com/uploadfile/pdf/0/0/239.pdf (accessed on 20 October 2022).
52. Nuclei N200 Series 32-Bit High Performance Processor. Available online: https://www.nucleisys.com/product/200.php (accessed on 20 October 2022).
53. ESP32-SA Series Datasheet. Available online: https://www.espressif.com/sites/default/files/documentation/esp32-s2_datasheet_en.pdf (accessed on 20 October 2022).
54. Kendryte K210 System-on-Chip (SoC). Available online: https://maixduino.sipeed.com/en/hardware/k210.html (accessed on 20 October 2022).
55. Face Detection K210 Model Generated by Nncase. Available online: https://github.com/kendryte/kendryte-standalone-demo/tree/develop/face_detect (accessed on 20 October 2022).
56. Google Groups—Rocket Chip for Embedded, FPGA Area. Available online: https://groups.google.com/a/groups.riscv.org/g/hw-dev/c/zZxy0iFzrvI/m/LVeFiK2vAQAJ (accessed on 20 October 2022).

57. Build FPGA Bitstream—64-Bit Sonic BOOM Cores. Available online: https://github.com/eugene-tarassov/vivado-risc-v#build-fpga-bitstream (accessed on 20 October 2022).
58. Gookyi, D.A.N.; Ryoo, K. Selecting a synthesizable RISC-V processor core for low-cost hardware devices. *J. Inf. Process. Syst.* **2019**, *15*, 1406–1421.
59. Li, B.; Zhang, X.; You, H.; Qi, Z.; Zhang, Y. Machine Learning Based Framework for Fast Resource Estimation of RTL Designs Targeting FPGAs. *ACM Trans. Des. Autom. Electron. Syst. (TODAES)* **2022**, *28*, 24. [CrossRef]
60. Apache TVM—An End to End Machine Learning Compiler Framework. Available online: https://tvm.apache.org/ (accessed on 20 October 2022).
61. Chen, T.; Moreau, T.; Jiang, Z.; Zheng, L.; Yan, E.; Shen, H.; Cowan, M.; Wang, L.; Hu, Y.; Ceze, L.; et al. TVM: An automated End-to-End optimizing compiler for deep learning. In Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18), Carlsbad, CA, USA, 8–10 October 2018; pp. 578–594.
62. Moreau, T.; Chen, T.; Vega, L.; Roesch, J.; Yan, E.; Zheng, L.; Fromm, J.; Jiang, Z.; Ceze, L.; Guestrin, C.; et al. A hardware–software blueprint for flexible deep learning specialization. *IEEE Micro* **2019**, *39*, 8–16. [CrossRef]
63. microTVM: TVM on Bare-Metal. Available online: https://tvm.apache.org/docs/topic/microtvm/index.html (accessed on 20 October 2022).
64. Prakash, S.; Callahan, T.; Bushagour, J.; Banbury, C.; Green, A.V.; Warden, P.; Ansell, T.; Reddi, V.J. Cfu playground: Full-stack open-source framework for tiny machine learning (tinyml) acceleration on fpgas. *arXiv* **2022**, arXiv:2201.01863.
65. The CFU Playground: Accelerate ML Models on FPGAs. Available online: https://cfu-playground.readthedocs.io/ (accessed on 20 October 2022).
66. Renode—A Virtual Development Tool for Multinode Embedded Networks. Available online: https://github.com/renode/renode (accessed on 20 October 2022).
67. Amid, A.; Biancolin, D.; Gonzalez, A.; Grubb, D.; Karandikar, S.; Liew, H.; Magyar, A.; Mao, H.; Ou, A.; Pemberton, N.; et al. Chipyard: Integrated Design, Simulation, and Implementation Framework for Custom SoCs. *IEEE Micro* **2020**, *40*, 10–21. [CrossRef]
68. Zaruba, F.; Benini, L. 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 Trans. Very Large Scale Integr. (VLSI) Syst.* **2019**, *27*, 2629–2640. [CrossRef]
69. Lee, Y.; Schmidt, C.; Ou, A.; Waterman, A.; Asanović, K. *The Hwacha Vector-Fetch Architecture Manual, Version 3.8.1*; Technical Report UCB/EECS-2015-262; EECS Department, University of California: Berkeley, CA, USA, 2015.
70. Izraelevitz, A.; Koenig, J.; Li, P.; Lin, R.; Wang, A.; Magyar, A.; Kim, D.; Schmidt, C.; Markley, C.; Lawson, J.; et al. Reusability is FIRRTL ground: Hardware construction languages, compiler frameworks, and transformations. In Proceedings of the 2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), Irvine, CA, USA, 13–16 November 2017; pp. 209–216. [CrossRef]
71. Karandikar, S.; Mao, H.; Kim, D.; Biancolin, D.; Amid, A.; Lee, D.; Pemberton, N.; Amaro, E.; Schmidt, C.; Chopra, A.; et al. FireSim: FPGA-accelerated cycle-exact scale-out system simulation in the public cloud. In Proceedings of the 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA), Los Angeles, CA, USA, 1–6 June 2018; pp. 29–42.
72. Pemberton, N.; Amid, A. FireMarshal: Making HW/SW Co-Design Reproducible and Reliable. In Proceedings of the 2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Stony Brook, NY, USA, 28–30 March 2021; pp. 299–309. [CrossRef]
73. Spike RISC-V ISA Simulator. Available online: https://github.com/riscv-software-src/riscv-isa-sim (accessed on 20 October 2022).
74. Mantovani, P.; Giri, D.; Di Guglielmo, G.; Piccolboni, L.; Zuckerman, J.; Cota, E.G.; Petracca, M.; Pilato, C.; Carloni, L.P. Agile SoC development with open ESP. In Proceedings of the 2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD), San Diego, CA, USA, 2–5 November 2020; pp. 1–9.
75. Fahim, F.; Hawks, B.; Herwig, C.; Hirschauer, J.; Jindariani, S.; Tran, N.; Carloni, L.P.; Di Guglielmo, G.; Harris, P.; Krupa, J.; et al. hls4ml: An open-source codesign workflow to empower scientific low-power machine learning devices. *arXiv* **2021**, arXiv:2103.05579.
76. NVIDIA Deep Learning Accelerator (NVDLA) Is a Free and Open Architecture. Available online: http://nvdla.org/ (accessed on 20 October 2022).
77. GreenSocs QBOX, a Solution for Co-Simulation with QEMU and SystemC. Available online: https://www.machineware.de/#qemu (accessed on 20 October 2022).
78. Gonzalez, A.; Zhao, J.; Korpan, B.; Genc, H.; Schmidt, C.; Wright, J.; Biswas, A.; Amid, A.; Sheikh, F.; Sorokin, A.; et al. A 16 mm^2 106.1 GOPS/W Heterogeneous RISC-V Multi-Core Multi-Accelerator SoC in Low-Power 22 nm FinFET. In Proceedings of the ESSCIRC 2021—IEEE 47th European Solid State Circuits Conference (ESSCIRC), Grenoble, France, 13–22 September 2021; pp. 259–262. [CrossRef]
79. Giri, D.; Chiu, K.L.; Eichler, G.; Mantovani, P.; Chandramoorth, N.; Carloni, L.P. Ariane+ NVDLA: Seamless third-party IP integration with ESP. In Proceedings of the Workshop on Computer Architecture Research with RISC-V (CARRV), Virtual Event, 29 May 2020 .

80. Feng, S.; Wu, J.; Zhou, S.; Li, R. The Implementation of LeNet-5 with NVDLA on RISC-V SoC. In Proceedings of the 2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS), Beijing, China, 18–20 October 2019; pp. 39–42. [CrossRef]
81. Assir, I.A.; Iskandarani, M.E.; Sandid, H.R.A.; Saghir, M.A. Arrow: A RISC-V Vector Accelerator for Machine Learning Inference. *arXiv* **2021**, arXiv:2107.07169.
82. Patsidis, K.; Nicopoulos, C.; Sirakoulis, G.C.; Dimitrakopoulos, G. RISC-V2: A Scalable RISC-V Vector Processor. In Proceedings of the 2020 IEEE International Symposium on Circuits and Systems (ISCAS), Seville, Spain, 12–14 October 2020; pp. 1–5. [CrossRef]
83. Schiavone, P.D.; Rossi, D.; Di Mauro, A.; Gürkaynak, F.K.; Saxe, T.; Wang, M.; Yap, K.C.; Benini, L. Arnold: An eFPGA-augmented RISC-V SoC for flexible and low-power IoT end nodes. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2021**, *29*, 677–690. [CrossRef]
84. Flamand, E.; Rossi, D.; Conti, F.; Loi, I.; Pullini, A.; Rotenberg, F.; Benini, L. GAP-8: A RISC-V SoC for AI at the Edge of the IoT. In Proceedings of the 2018 IEEE 29th International Conference on Application-Specific Systems, Architectures and Processors (ASAP), Milan, Italy, 10–12 July 2018; pp. 1–4. [CrossRef]