CHALLENGES FOR PORTING A SPACE APPLICATION FROM ARM TO RISC-V ARCHITECTURE

K. Schwenk*, D. Herschmann*
* German Aerospace Center (DLR), German Space Operations Center (GSOC), Münchner Str. 20, 82234 Wessling, Germany

Abstract

RISC-V, an open and royalty-free instruction set architecture that standardizes processor-software interaction, is increasingly adopted in spaceborne embedded systems due to its customizability and vendor-neutral design. In recent years, several on-board computer architectures leveraging this ISA have been proposed by major space agencies and industry providers, including NASA. This work addresses the feasibility of migrating existing space applications to RISC-V-based platforms by documenting the porting process of ODARIS, a medium-sized C++ application (~50K LOC) with dependencies including Boost, OpenCV, and TensorFlow Lite, from ARM/x86 to a Microchip PolarFire SoC FPGA featuring RISC-V cores. The study evaluates the maturity of the RISC-V Linux ecosystem, assesses dependency availability, and benchmarks computational performance against established platforms such as the AMD Zynq 7000 and NVIDIA Jetson Orin. Our analysis outlines the technical steps and challenges encountered during cross-compilation and runtime adaptation, and presents quantitative performance comparisons across the architectures. The findings indicate that while the RISC-V software ecosystem is maturing, successful migration requires careful dependency management and thorough performance validation. These insights provide a practical effort assessment for developers evaluating RISC-V adoption in space applications, supporting informed architectural decisions for next-generation on-board computers.

Keywords

real-time communication, software development, embedded systems, satellite platform

1. INTRODUCTION

> The "instruction set architecture" (ISA) is part of the abstract model of a computer that defines how the Central processing unit (CPU) is controlled by the software. The ISA acts as an interface between the hardware and the software, specifying both what the processor is capable of doing as well as how it gets done. [1]

The reduced instruction set computer five (RISC-V) ISA is a fifth-generation "reduced instruction set computer" (RISC) architecture developed at the University of California, Berkeley [2]. The first user manual for the ISA was published in 2011 as a technical report [3]. The primary motivation for its development was to provide an accessible open standard that enables users to freely implement, modify, and share designs. It was not developed to mitigate the technical limitations of existing proprietary ISAs, such as the widely used x86 and ARM architectures. The use of these proprietary ISAss is heavily restricted by their license holders, including substantial licensing fees in the millions of dollars, restrictions on developing custom implementations, and prohibitions against customizing the ISA [4]. As an example, there are currently only two companies offering modern x86 implementations, namely Intel and AMD, and only a few large companies, such as Apple, hold licenses to create their own ARM designs. To direct the development and drive the initial adoption of the RISC-V ISA the RISC-V foundation was founded in 2015. In 2018 the RISC-V foundation announced a joint collaboration with the Linux Foundation to provide operational, technical and strategic support. To circumvent any possible U.S.-related export restrictions, the RISC-V foundation relocated to Switzerland to provide legal protection for the RISC-V community and its ecosystem investments [2]. Currently the RISC-V foundation has 271 members, ranging from large companies such as NVIDIA to universities such as the ETH Zürich and space-related organizations such as the ESA [5].

The RISC-V architecture is already commercially available. As an example, NVIDIA uses RISC-V-based microcontrollers in its graphic processing unit (GPU) products and has stated that it has already shipped billions of RISC-V cores [6]. NASA based its next high-performance spaceflight computer platform on the RISC-V architecture [7]. Companies like SiFive [8] and Microchip [9] offer RISC-V CPU designs and implementations ranging from microcontrollers to servers. Even consumer-grade hardware, such as

CC BY 4.0

doi: 10.25967/680234

the Raspberry Pi Pico 2 RISC-V microcontrollers, is available for educational and home use [10].
The RISC-V architecture also benefits from substantial software platform support. Several Linux distributions are available, such as Debian [11], Ubuntu [12], and openSUSE [13]. Additionally, common real-time operating systems such as RTEMS [14] and FreeRTOS [15] offer RISC-V support.
The goal of this paper to provide an overview of the current state of the RISC-V ecosystem by presenting background information and a sharing experience migrating a medium-sized space application to the RISC-V architecture. These insights are intended to help developers gain an understanding of the RISC-V ecosystem and enable them to perform an initial assessment of the feasibility of adopting RISC-V for their next space application projects.
In the second chapter, a brief introduction of the RISC-V ISA is presented. In the third and fourth chapter, an overview of currently available hardware and software platform options are presented. In the fifth chapter an experience report on porting an application to the RISC-V architecture is provided, followed by a short conclusion in the sixth chapter.

2. RISC-V ISA

In this chapter, a brief description of the RISC-V ISA is provided. First, the concept of an ISA is clarified. An ISA serves as an abstract contract between a processor's hardware and the software it executes. It defines, on one hand, how a programmer must write a correct program and on the other hand how the CPU must interpret and execute the program. An ISA typically specifies the supported machine instructions, data types, registers, memory access models and I/O handling [1]. Software developed according to an ISA must be executable on any CPU that implements the ISA. Derivations from the ISA specification lead to discrepancies between the intended and actual behaviour of a program. While such discrepancies most commonly arise from software errors, they can also result from hardware implementation flaws.
The RISC-V ISA is published by the RISC-V Foundation under the "Creative Commons Attribution 4.0 International License". It can be freely accessed in PDF or HTML format on the foundation's website without registration [2]. As the RISC-V ISA aims to cover a broad range of applications ranging from tiny educational microcontrollers to high-performance CPUs for supercomputers, it was strictly modularized and designed to be highly configurable.
The RISC-V ISA documentation is divided into two classes: the privileged and unprivileged ISA. The privileged ISA describes the instruction set components relevant for Operating System (OS) level development, whereas the privileged ISA describes those relevant for general application programming. The unprivileged part will not be discussed in this paper, but when choosing a system, it is advisable to verify the documentation for compatibility with the target OS or bare-metal runtime environment.
The following section provides an overview of the unprivileged ISA, as it may be helpful for an application programmer to understand the platform's capabilities. It can also aid in configuring the software build environment to generate software that executes optimally on the target platform while fully utilizing available resources. The RISC-V unprivileged ISA configuration is defined by a base layer, which establishes the general system layout, and by optional extension layers that add specific features. The currently ratified base layers are listed in Table 1. Base layers specify whether the architecture uses a 32-bit or 64-bit register width. In addition to the default instruction set, a version targeting embedded systems is also available. The following section provides an overview of the unprivileged ISA, as it may be helpful for an application programmer to understand the platform's capabilities. It can also aid in configuring the software build environment to generate software that executes optimally on the target platform while fully utilizing available resources. The RISC-V unprivileged ISA configuration is defined by a base layer, which establishes the general system layout, and by optional extension layers that add specific features. The currently ratified base layers are listed in Table 1. Base layers specify whether the architecture uses a 32-bit or 64-bit register width. In addition to the default instruction set, a version targeting embedded systems is also available. The primary distinction of the embedded ISA is that it provides half the number of registers to enable smaller CPU designs. Some of the most important extensions are listed in Table 2. To precisely describe the configuration of an unprivileged RISC-V ISA, an extension naming convention is used, which concatenates the base layer and extension identifiers sequentially. A typical example is RV64IGC. Further details regarding the naming convention can be found in the ISA Extension Naming Convention section [16]. Additional properties of the RISC-V ISA are summarized in Table 3. The primary distinction of the embedded ISA is that it provides half the number of registers to enable smaller CPU designs. Some of the most important extensions are listed in Table 2. To precisely describe the configuration of an unprivileged RISC-V ISA, an extension naming convention is used, which concatenates the base layer and extension identifiers sequentially. A typical example is RV64IGC. Further details regarding the naming convention can be found in the ISA Extension Naming Convention section [16]. Additional properties of the RISC-V ISA are summarized in Table 3.
The RISC-V ISA enables the construction of highly optimized instruction sets tailored to specific applications, but this flexibility results in a combinatorial explosion of possible ISA configurations. To prevent ecosystem fragmentation, ISA profiles for common use cases were defined. Table 4 lists the currently defined profiles. Different profile versions are identified by the release year in their identifier.

CC BY 4.0

Name	Description
RV32I	Base integer instruction set, 32-bit
RV32E	Base integer instruction set (embedded)
RV64I	Base integer instruction set, 64-bit
RV64E	Base integer instruction set (embedded)

TAB 1. RISC-V base layers (unprivileged)

Name	Description
M	Integer multiplication and division
A	Atomic instructions
F	Single-precision floating-point
D	Double-precision floating-point
Zicsr	Ctrl/Status register (CSR)
Zifencei	Instruction-fetch fence
G	Shorthand for IMAFD_Zicsr_Zifencei
B	Bit manipulation instructions
C	Compressed instructions
V	Vector operations
H	Hypervisor support

TAB 2. RISC-V extension layers (unprivileged)

Name	Description
General purpose registers	32 (RV[32|64]E 16)
Floating point registers*	32 (RV[32|64]E 16)
Instruction width	32 bit (comp* 16 bit)
Byte encoding instructions	le (little endian)
Byte encoding data**	le || be || bi
Integer encoding	2 complement
FP encoding*	IEEE 754-2008
Exec modes*	U|S|R|M ***

TAB 3. RISC-V ISA general properties (* if supported by extensions, ** configuration dependent, * unsupervised, supervised, reserved, machine mode)**

The RVU32/RVU64 profiles serve as a baseline for providing a generic target for 32/64-bit software toolchains and represent a minimum compatibility level. They require RV[32|64]I support and mandate little-endian byte encoding. The RVA[20/22/23] profiles are intended to establish a common standard for general-purpose application processors capable of running general-purpose operating systems and applications. Typical use cases include servers, desktops, or mobile systems running standard Linux operating system (Linux) distributions. These profiles support all commonly used unprivileged extensions, including RV64IGC. Starting with RVA23, the vector extension becomes mandatory, bringing the ISA to a performance level comparable to currently deployed x86 and ARM ISAs. The RVA23 standard is expected to become the minimum requirement for running future default Linux kernels; for instance, Ubuntu already requires RVA23 for releases 25.10 and later [12]. The application processors in NASA's next high-performance spaceflight computer (HSPC) support the RVA23 profile [7]. However, aside from this, most other RISC-V systems currently available for space avionics only support RV[32|64]GC. The RVB profile is intended as a baseline for specialized application processors targeting individual software ecosystems. A compact overview of the RISC-V ISA, including base and extension layers, historical context, and profiles, can be found on the Wikipedia RISC-V page [17].

Name	Description
RVI20U[32|64]	Minimal compatibility profile
RVA[20|22|23]	Application processor (ap)
RVB[23]	Customizable ap baseline

TAB 4. RISC-V profiles

3. RISC-V SPACE HARDWARE OPTIONS

The following section provides a short overview of RISC-V hardware platforms suitable for space applications. A list of currently available options is shown in table 5.

One of the most promising RISC-V platforms for space is the Microchip PIC64 HPSC platform which is the basis for NASA's planned next-generation high performance spaceflight computing (HSPC) platform [7]. As described in the HSPC white paper the HSPC computer platform [18] was designed for autonomy and AI at the edge of space and the demanding fault-tolerance requirements of human space exploration and operations missions [18]. The HSPC contains eight RVA23-compliant SiFive X280 RISC-V cores. To support AI-processing, the X280 core implements the RISC-V Vector Extension (RVV) with a 512-bit vector register width. When extended reliability is required the X280 RISC-V cores can be paired in a lock-step

CC BY 4.0

mode, where the CPUs process the same input and when the output differs failure handling routines are triggered. Evaluation boards for the PIC64-HPSC are currently available on the market. The Frontgrade Gaisler GR801 is a competing European solution. Unlike the PIC64 HSPC it is currently in the development phase. It is also designed for AI applications, including image recognition, autonomous navigation, and data analysis. In contrast to the PIC64 HSPC, an accelerator architecture approach is chosen. The GR801 SOC consists of a single NOEL-V FT RISC-V RV64GC processor for system management tasks. For AI computation an Akida accelerator is integrated that supports AI-frameworks such as TensorFlow and Keras [19]. Frontgrade Gaisler also develops the radiation-tolerant GR765 FPGA SoC, which offers eight LEON5FT SPARC V8 cores and 8 NOEL-V 64 RISC-V RV64GCH cores. At boot time the system can select between the SPARC or RISC-V cores. It also contains a programmable FPGA section [20]. A competitor for the GR765 is the Microchip PolarFire SoC FPGA which contains four RV64GC cores. Unlike the Gaisler products, PolarFire development boards are already commercially available.

In terms of performance, the Gaisler GR801, GR765, and PolarFire SoCs belong to a similar performance class. They are all RV64GC dual-issue in-order CPU designs [21]. The SiFive X280 core in the PIC64-HSPC is also a dual-issue in-order CPU design but adds compatibility to the new RVA23 standard, which makes it future-proof for new software standards, and also offers vector processing capability. Performance-wise it is comparable to ARM Cortex-A9 based processors, such as those integrated into the Zynq-7000 SoC platform [22]. The ARM Cortex-A9 is also an in-order design with an additional NEON SIMD-vector unit. Compared to x86 designs, they are similar to a P5-Pentium design from 1993, but much higher clock speeds and with additional vector capabilities. The RISC-V cores are outperformed by more recent ARMv8/v9 designs such the Cortex-A78, which is integrated into the Nvidia Jetson Orin-series SoCs. Their cores feature a powerful out-of-order designs, advanced vector capabilities and substantially higher clock rates [23]. Furthermore, the Jetson platform includes integrated GPUs that far exceed the computational capabilities of the evaluated RISC-V systems. Computational performance is still a significant trade-off in the field of space-graded hardware. This section provides a high-level classification. A comprehensive performance evaluation would require dedicated benchmarking and testing.

4. SOFTWARE ECOSYSTEM

In this section, a brief superficial overview of the current software ecosystem surrounding RISC-V is provided. This overview is meant to give the reader a general understanding of the software tools currently available for the RISC-V platform and to provide references where more detailed information can be obtained if required. The information presented has been collected through a literature review only, without empirical verification by the authors, which is beyond the scope of this article. Nevertheless, a hands-on report of s specific platform is provided in the next sections.

Name	Description
Spacegrade Grade CPU's	
Microchip PIC64 HPSC RH/RT	8*RVA23 [24]
Frontgrade Gaisler GR801	1*RV64GC [19]
Rad-tol FPGA's SOCs	
Microchip PolarFire SoC FPGA	4*RV64GC [20]
Frontgrade Gaisler GR765	8*RV64GCH [25]

TAB 5. RISC-V hardware spaceflight grade

Overall, 16 years after the first publication of the RISC-V ISA in 2010, a rich ecosystem around RISC-V has been established. Important OS, tools and libraries used for embedded system development are now available for RISC-V platforms. A summary can be seen in table 6.

For real-time operating system (RTOS) there is rich support for systems commonly used in the space sector. Recent versions of the Real-Time Executive for Multiprocessor Systems (RTEMS) have support for RISC-V on various hardware platforms [14]. Minimal configurations of RTEMS require only RV32I support, making them compatible with hardware platforms such as the Microchip Polar Fire SoC, NOEL-V systems from Gaisler, and microcontrollers like the Espressif Systems ESP32-C3 [26] and more. Free Real Time Operating System (FreeRTOS) also offers RISC-V support for selected platforms such as the PolarFire Soc and the MiFive M2GL025 Creative Board, but far less than RTEMS [15]. For example, support for Gaisler systems seems to be lacking. The Zephyr RTOS platform from the Linux Foundation also offers support for various systems such as the Microchip PolarFire Soc and the Gaisler NOLE-V architecture [27]. This list presents only a subset of available RTOS systems. Other commercially available systems such as VxWorks, ROS, and Wind River may also support specific platforms. Especially Microchip PolarFire Soc offers a broad range of operating system support. However, RTOS RISC-V support is highly platform-specific, and it has to be verified which OS version and configuration is supported for a given target platform. Especially Microchip PolarFire Soc offers a broad range of operating system support. But RTOS RISC-V support is platform-specific and it has to be checked which version of an operating system are supported in what configuration.

The RISC-V platform also benefits from good Linux support. Binary Linux distributions such as Debian [11] (since 2023) and Ubuntu [12] are available and provide out-of-the-box solutions. For specific

CC BY 4.0

platforms and configurations, highly customized Linux images can be built using the embedded-focused Yocto-Linux source project [28], although it requires a high level of expertise in setting up such a build environment and configuration, which may not be feasible for smaller teams. A minimum standard for running a Linux distribution appears to be an RV64GC-compatible RISC-V system [11]. Furthermore, most of the Linux distribution packages are compatible with the RISC-V ISA. Of the more than 30000 packages provided by Debian, only 95 currently fail to be built [29].

Linux support for RISC-V remains platform-specific for now, as one specific distribution may only be available for one specific board. For example, Ubuntu 24.04.4 LTS is available for the Microchip PolarFire SoC FPGA Icicle boards but not for the PolarFire Discovery Kit. A current limitation is that, despite Linux having no generic support for the many currently available RISC-V systems due to the absence of a unified hardware baseline [30]. Vendors often provide out-of-tree Linux kernel forks with many custom patches and drivers included to ensure compatibility with their products. Consequently, users are often stuck to specific kernel versions. However, the RVA23 profile which was ratified in 2024, appears to become a de facto standard for mainline Linux kernel support [30]. For example, RVA23 will be required for Ubuntu versions 25.10 and upwards. Nevertheless, most currently available RISC-V CPUs do have RVA23 support, such as the Microchip PolarFire SoC or the still-to-be-released Gaisler systems.

Additionally, most of the tools required for developing software for RISC-V systems are available. Both major open-source build toolchains, GCC and LLVM, provide (cross-)compilation support for RISC-V targets, along with debugging and profiling tools [31,32]. The QEMU software emulator enables the execution of RISC-V applications and complete OS instances on x86 and ARM systems. Java applications can be executed on RISC-V systems with the OpenJDK (OpenJava Development Kit), which includes native support. Many important software libraries, such as OpenCV [29], are available for RISC-V platforms.

In summary, the literature research indicates that tools for embedded development and onboard space computing are generally available for the RISC-V platform. Nevertheless, software support remains heavily dependent on the specific hardware platform and careful selection if required for each project.

Name	Description
RT-OS	Rtems [14], FreeRtos [15], Zephyr [27]
LinuxBin	Debian [11], Ubuntu [12], OpenSuse [13]
LinuxSrc	Yocto [28]
Tools	openjdk, v8, gcc [31], llvm [32], qemu [33]
Libs	opencv [34], boost [35]

TAB 6. RISC-V ecosystem support

5. EXPERIENCE REPORT

Within this section, a hands-on experience report on deploying On-board Data Analysis And Real-Time Information System (ODARIS) to a RISC-V platform shall be provided. ODARIS is an information system, designed for enabling real-time (RT) telemetry for satellite systems. It is currently classified as experimental software and shall be tested on the Captn-1 (6-U CubeSat) [36] and SeRANIS (Compact satellite) [37] missions. It includes a broad set of components performing on-board data analysis, data management, and communication services. General information about the application is listed in Table 7. More information about ODARIS can be found in [38], and details about the AI-based data analysis are provided in [39].

From a technical perspective, ODARIS is a Linux-based C++ programming language (C++) application. It consists of several components that depend on widely used C++ libraries such as C++ Boost [40], GSL, OpenCv [41], SQLite [42], and TFLite [43]. It currently offers platform support for x86_64 (Debian), ARMv7a (Zynq7000/Poky v4.0) and ARMv8 (NVIDIA Jetson platform/Debian) architectures.

Name	Description
Status	Experimental system
Size	1225 files, 420K Loc
Lang*	C++20 (78%), Python3 (8%)
Build	CMake with GCC & Clang
Libs*	Boost, OpenCV, SQLite, GSL
OS	Linux
Plat	x86_64, ARMv7, ARMv8
Mission	CAPTn-1, SeRANIS (both 2026/2027)
TMTC	ECSS-PUS-C, text-stream

TAB 7. Overview of the ODARIS space application used as an case example, * most important items

For analysis of the compatibility for potential future RISC-V-based on-board computing systems, the hardware target the PolarFire SoC Discovery Kit platform [44] was selected. The discovery kit includes a development board that is based on a PolarFire SoC FPGA with four U54 cores (RV64GC), 1 GByte of DDR4-RAM, a microSD card slot, a single 1 GBit Ethernet port, and a USB type-C port used for power supply and serial access. Based on the official documentation, it also provides Linux support [44].

The main reasons for choosing the Microchip PolarFire SoC FPGA as a hardware platform were that, it is a potential candidate for a next-generation DLR on-board computing system, the development board offers all necessary interfaces for rapid development without the need to provide additional hardware, it has Linux support, and finally the hardware is currently available at a reasonable price (USD: 150$).

CC BY 4.0

After acquisition, the first step is to install a functional Linux system on the embedded device and establish minimal terminal access from the developer host system. Based on the Discovery Kit Quickstart Guide, references to a Poky Linux-based reference design firmware is provided [45] along with a pre-built release version [46]. Otherwise, the firmware has to be generated by the Microchip FPGA toolchain, which requires significant time to get familiar with, until the generation of a suitable configuration is possible. Nevertheless, an additional vendor-specific flash program is required from the company website. The software supports only Windows and a specific Ubuntu distribution, which becomes challenging when working on a Debian-based Linux system. To circumvent this incompatibility, a virtual machine or lightweight Docker container, with the specific Ubuntu version installed, can be used to re-flash the device. The firmware contains a SoC reference configuration, which is suitable for running Linux operating systems, along with a bootloader required to load the OS. As a bootloader, the open-source U-Boot [47] is available, which is a quasi standard in the embedded world. After the boot loader is installed, the Discovery Kit can be accessed via the serial interface. However, no Linux OS is installed yet on the board. As an out-of-the-box available Linux system, only a specialized minimal PolarFire SoC embedded Yocto Poky Linux (Kirkstone 4.0.17) is available. The root image v2024.06 from the official PolarFire SoC yocto release page [48] has to be flashed on a microSD card. Network connectivity, including Secure Shell (ssh) access can be established using standard Linux network configuration.

The minimal Yocto-Poky Linux installation is not suitable to run complex application software, such as ODARIS, without installing additional required runtime library dependencies and its build environment. One option is to create a customized Yocto-Poky image, based on the available platform support layer provided by Microchip. Building a Yocto image requires creating a list of all required packages (recipes), choosing compatible versions, and patching minor incompatibilities. For more complex libraries such as OpenCV and TensorFlow, it is often necessary to perform customization of the build configuration, select compatible source versions, and sometimes apply source patches within the dependency source code. Due to limited resources, this option is often not feasible in smaller development teams and also not performed in the context of this study. Nevertheless, Yocto-Poky offers the means to build a highly tailored configuration that would can't be offered by a standard binary distribution.

Another option is to find a suitable binary distribution compatible with the used PolarFire SoC development kit. In context of this kit, no suitable distribution could be found for now, but Ubuntu 24.04.04 LTS version is available for the Microchip PolarFire SoC FPGA Icicle. The Icicle kit is a higher-tier development board, but it is based on the same SoC. Given that the Linux kernel is designed to offer a stable ABI interface, a combination of the Ubuntu 24.04.04 LTS root image together with an already working Poky boot image, providing a compatible kernel, may be feasible. This creates a Ubuntu 24.04.04 LTS with a customized kernel. With additional effort it should also be possible to replace the kernel or exchange the device tree of the Ubuntu 24.04.04 LTS image with one that is compatible with the board. Once running Ubuntu 24.04.04 LTS and enabled network support, it is possible to install all tools and libraries required to build the basic components of ODARIS on the PolarFire SoC FPGA. Nevertheless, long build times for native compilation of a large software packages have to be considered. Compilation of the basic ODARIS components on an recent x86_64 CPU takes around 5-10 minutes, while on such an embedded device, it takes up to several hours. The final bottleneck is the available system memory of around 600 MB , which may be insufficient to build larger components.

Due to the memory limitation, compilation of larger software components requires a cross-compile build environment on a more powerful system. A typical Linux Distribution, such as Debian, provides all required tools like a RISC-V-compatible GCC compiler suite, containing all tools for building, and can be installed via standard package managers. The provision of a RISC-V system root image is more complex. For users unfamiliar with the term, a brief explanation of a system root image is provided below. Building C++ applications requires availability of necessary libraries (.so/.a) files, compatible header files (interface files), and configuration files (CMake files) available at build time. In native builds, these files are installed in standard include or library directories, enabling the developer to compile software compatible with the developer host architecture. In most cases, developers can simply use the distribution package manger to install these resources. When targeting other architectures, however, using the default package manager is not an option, as it typically does not support the target platform. For Linux builds, the standard approach is to provide a complete target filesystem. This is typically part of the Yocto-Poky build environment development package. For binary Linux Ubuntu distributions, a different approach can be chosen. Linux offers methods to dynamically switch the root filesystem (chroot). Standard system roots for specific distributions are available and contain the package manager, which can be used to install additional required packages. To run programs within the RISC-V system root QEMU can be used as middleware system. This approach enables the creation of a system root directory suitable for building most of the ODARIS components. Components such as TFLite can't be built out-of-the-box and needs additional effort in patching build system within the dependency library.

In summary, it could be verified, that a complete Ubuntu 24.04.04 LTS environment can be installed and utilized on the PolarFire SoC development Kit, with nearly all standard Ubuntu packages available as

CC BY 4.0

well as a complete cross-compile build environment for C++ targets which only needs minor adjustments. More specialized libraries such as TFLite are not available out-of-the-box but can likely be installed on the system with additional effort. These findings confirm the statements in the literature's assessment that RISC-V enjoys robust ecosystem support, though careful platform selection remains essential.

6. CONCLUSION AND WAY FORWARD

This article provided a concise introduction to the RISC-V ecosystem, covering the essential aspects required to understand and deploy software on a RISC-V platform. An overview of the RISC-V ISA specification was presented to enable developers in selecting compatible hardware and software platforms. The paper reviews currently available RISC-V hardware suitable for space applications and examines the corresponding software ecosystems, highlighting platform options for future space systems. The final chapter presented a hands-on experience report on establishing a RISC-V build environment. Within the scope of this article, certain components of the target space application could not be compiled due to missing dependencies and resources. Nevertheless, the findings indicate that despite RISC-V being a relatively new architecture, it already benefits from major platform support. Currently, most available hardware platforms offer out-of-the-box support for specific Linux and RTOS platforms. Space-qualified systems currently support standard Linux distributions such as Ubuntu 24.04.04 LTS and RTEMS, with the majority of software packages available. Successful system selection and the configuration of both development and runtime environments require a fundamental understanding of the underlying technology stack, which this article aimed to provide.

Contact address:

kurt.schwenk@dlr.de

References

[1] Arm Limited. What is instruction set architecture. https://www.arm.com/glossary/isa.

[2] RISC-V International. Risc-v foundation, 2026. https://riscv.org/.

[3] Andrew Waterman; Yunsup Lee; David A. Patterson; Krste Asanovic. The risc-v instruction set manual, volume i: Baseuser-level. techreport, Electrical Engineering and Computer SciencesUniversity of California at Berkeley.

[4] David A. Patterson Krste Asanović. Instruction sets should be free: The case for risc-v. Technical Report Technical Report No. UCB/EECS-2014-146, Electrical Engineering and Computer SciencesUniversity of California at Berkeley.

[5] RISC-V international. Risc-v international list of members. https://riscv.org/members/.

[6] Anshia Sharma (RISC-V international). How nvidia shipped one billion risc-v cores in 2024. https://riscv.org/blog/how-nvidia-shipped-one-billion-risc-v-cores-in-2024/.

[7] NASA. High performance spaceflight comptuing, 2026. https://www.nasa.gov/game-changing-development-projects/high-performance-spaceflight-computing-hpsc.

[8] Inc. SiFive. Sifive company. https://www.sifive.com/.

[9] Mircoship Technology Inc. Microship technolgoy inc. https://www.microchip.com/.

[10] Raspberry Pi Foundation. Rasperry pi pico 2. https://www.raspberrypi.com/products/raspberry-pi-pico-2/.

[11] Debian project. Risc-v. https://wiki.debian.org/RISC-V.

[12] Canonical Ltd. Canonical ubuntu risc-v. https://ubuntu.com/download/risc-v/canonical-built.

[13] SUSE LLC. opensuse risc-v. https://en.opensuse.org/openSUSE:RISC-V.

[14] RTEMS-Project. Risc-v manual, section 8.1.1: Board support packages for riscv. https://docs.rtems.org/docs/main/user/bsps/bsps-riscv.html.

[15] Amazon Web Services. Using freertos on risc-v microcontrollers. https://www.freertos.org/Using-FreeRTOS-on-RISC-V.

CC BY 4.0

[16] RISCV international. Isa extension naming conventions. https://docs.riscv.org/reference/isa/v20260120/unpriv/naming.html.

[17] Wikipedia. Risc-v. https://en.wikipedia.org/wiki/RISC-V.

[18] NASA. White paper- high performance space flight computer (hspc). https://www.nasa.gov/wp-content/uploads/2024/07/hpsc-white-paper-tmg-23jul2024.pdf?emrc=6a850135d12a6.

[19] Frontgrade Gailser. Gr801 product page. https://www.gaisler.com/products/gr801.

[20] Microship Technology Inc. Productpage polarfire soc fpga series. https://www.microchip.com/en-us/products/fpgas-and-plds/system-on-chip-fpgas/polarfire-soc-fpgas.

[21] Frontgrade Gaisler. Noel-v product page. https://www.gaisler.com/products/noel-v.

[22] Advanced Micro Devices, Inc. Zynq-7000 soc product description, 2020. https://www.amd.com/de/products/adaptive-socs-and-fpgas/soc/zynq-7000.html.

[23] NVIDIA Corporation. Nvidia jetson modules website, 2023. https://developer.nvidia.com/embedded/jetson-modules.

[24] Microchip Technology Inc. Pic64-hpsc high-performance spaceflight computing series. https://www.microchip.com/en-us/products/microprocessors/64-bit-mpus/pic64-hpsc.

[25] Frontgrade Gaisler. Gr765 product page. https://www.gaisler.com/products/gr765.

[26] Espressif Systems. Esp32-c product page. https://www.espressif.com/en/products/socs/esp32-c3.

[27] Zephyr-Project. Zephyr support status on risc-v processors. https://docs.zephyrproject.org/latest/hardware/arch/risc-v.html.

[28] Trevor Gamblin (RISE). Blog: Improving risc-v support in the yocto project. https://riseproject.dev/2026/06/05/improving-risc-v-support-in-the-yocto-project/.

[29] Debian community. Ftbfs packages. https://udd.debian.org/cgi-bin/ftbfs.cgi?arch=riscv64.

[30] James De Vile (RISCV-International). Blog: Full-fat, kernel-ready: Why risc-v linux needs everyone upstream. https://riscv.org/blog/risc-v-upstreaming/.

[31] Linux Foundation. Risc-v collaboration. https://github.com/riscv-collab.

[32] LLVM-Project. User guide for risc-v target. https://llvm.org/docs/RISCVUsage.html.

[33] QemuProjectDevelopers. Risc-v system emulator. https://www.qemu.org/docs/master/system/target-riscv.html.

[34] OpenCV Team. Opencv risc v. https://github.com/opencv/opencv/wiki/OpenCV-RISC-V.

[35] EESSI-project. Eessi-index-boost. https://www.eessi.io/docs/available_software_riscv/detail/Boost/.

[36] DLR e.V. Captn-1 cubesat-mission. https://www.dlr.de/en/research-and-transfer/space/strategy-and-programme/space-system-technology/captn-1-cubesat-mission.

[37] SeRANIS Redaktions-Team. Seranis public website, 2023. https://seranis.de/.

[38] Kurt Schwenk and Daniel Herschmann. On-board data analysis and realtime information system - status & outlook. In *Deutscher Luft- und Raumfahrtkongress 2022 (DLRK 2022)*, September 2022.

[39] Mattia Scaglioni, Daniel Herschmann, and Kurt Schwenk. Yoshi: Yolo-odaris ship-detection interface - an innovative approach to on-board data analysis for ship-detection. In *European Data Handling & Data Processing Conference 2025 (EDHPC 2025)*, Oktober 2025.

[40] Boost. Boost c++ libraries project homepage, 2022. www.boost.org.

[41] OpenCv Team. Opencv project homepage, 2022. https://opencv.org/.

[42] SQLite Project. Sqlite project homepage, 2023. https://sqlite.org/.

[43] Google Inc. Tensorflowlite project homepage, 2023. https://www.tensorflow.org/lite.

[44] Microchip Technology Inc. Polarfire® soc discovery kit. https://www.microchip.com/en-us/development-tool/mpfs-disco-kit.

[45] Microship Inc. Polarfire® soc discovery kit reference design generation tcl scripts - libero® soc v2025.2+. https://github.com/polarfire-soc/polarfire-soc-discovery-kit-reference-design.

[46] Microship Inc. Discovery kit reference design release 2024.06. https://github.com/polarfire-soc/polarfire-soc-discovery-kit-reference-design/releases/tag/v2024.06.

[47] U-Boot development community. Das u-boot (the universal boot loader). https://u-boot-project.org/.

[48] Microship Inc. meta-polarfire-soc-yocto-bsp v2024.06. https://github.com/polarfire-soc/meta-polarfire-soc-yocto-bsp/releases/tag/v2024.06.

CC BY 4.0