Porting OpenEMS and Apache StreamPipes energy management software to freshly developed RISC-V system software stacks

Holger Blasum[1], Darshak Sheladiya[1], Jan Reinhard[1], Florian Krebs[1], David Engraf[1], Samuel Ardaya-Lieb[2], Frederik Haxel[3], Dominik Riemer[4], George Suciu[5], Mari-Anais Sachian[5], Robert Florescu[5]

[1] *SYSGO GmbH, Am Pfaffenstein 8, 55270 Klein-Winternheim, Germany*
{holger.blasum,darshak.sheladiya,jan.reinhard,florian.krebs,david.engraf}@sysgo.com
[2] *Consolinno Energy GmbH, Franz-Mayer-Str. 1, 93053 Regensburg, Germany*
s.ardayalieb@consolinno.de
[3] *FZI Research Center for Information Technology, Haid-und-Neu-Str. 10-14, 76131 Karlsruhe, Germany*
haxel@fzi.de
[4] *Bytefabrik.AI GmbH, Haid-und-Neu-Str. 10-14, 76131 Karlsruhe, Germany*
dominik.riemer@bytefabrik.ai
[5] *BEIA, Peroni 16, 041386 Bucharest, Romania*
{george,anais.sachian,robert.florescu}@beia.ro

Abstract

We port OpenEMS and an Apache StreamPipes extension service to RISC-V, validating the applications and system software stacks (Java, Linux, hypervisor, and Zephyr) beneath. We describe our hands-on experience in running these different payloads.

1. Introduction

The RISC-V ecosystem comprises both hardware and software. Having a rich software system, consisting of both system software (hypervisor, OS, virtual machines) as well as demonstrated applications is beneficial. In particular, RISC-V is relevant for safe and secure hypervisors and operating systems, as the RISC-V platform allows for both custom and open designs (e.g., analyzable for side channels). In the ISOLDE project [17], we validate the viability of RISC-V for energy management and smart home, using OpenEMS and Apache StreamPipes as applications, and also validating ports of a hypervisor and an embedded Linux, aligned with a general approach in the TRISTAN [28] and ISOLDE projects to have demonstrators in several industrial application domains [7].

We iterated incrementally, starting with the RISC-V QEMU emulator (Section 3). Then we used this environment to test an OpenEMS port for RISC-V (Section 4), validating the usability of the adaptation. Next, we replaced the QEMU emulator by the RISC-V CVA6 core running on a Genesys2 board (Section 5), using the FPGA platform to test and validate the latest version of the CVA6 core. One of the main targets of the TRISTAN/ISOLDE ecosystem. Then we replace QEMU with a hypervisor (Section 7), validating the hypervisor's functionality. Section 8 features an Apache StreamPipes extension service on a Banana Pi device, demonstrating the HW/SW stack on a commercial off-the-shelf embedded platform. We describe our hands-on experience in running these payloads.

Holger Blasum et al.. This is an open-access article distributed under the terms of the Creative Commons Attribution 4.0 License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

2. Related open-source work in energy

Open-source Energy Management Systems (EMS) are increasingly relying on commodity and industrial single-board computers (SBCs) and open CPU architectures to realize cost-effective deployments in smart homes and microgrids. Below, we review relevant hardware platforms (boards and chips) used as EMS edge controllers and gateways.

OpenEMS, Apache StreamPipes, and typical edge hardware. OpenEMS is a modular, open-source EMS running on-site (edge) to interface with devices (e.g., PV inverters, batteries) via industrial protocols [13, 12]. Apache StreamPipes is a distributed Industrial IoT analytics framework for ingesting and analyzing streaming data from industrial devices. In practice, both OpenEMS Edge and Apache StreamPipes (with its edge extension service) are commonly deployed on SBCs:

i. *Raspberry Pi* for prototyping and small commercial pilots owing to cost and sufficient I/O to reach smart meters and inverters [8, 13]. Version 5 has a Broadcom BCM2712 quad-core ARM Cortex-A76 @ 2.4GHz and LPDDR4X-4267 RAM (options for 1/2/4/8/16 GB) [24].
ii. *Banana Pi* class devices are used as low-cost Linux SBCs with improved I/O; within ISOLDE, Banana Pi hosts data services (e.g., Apache StreamPipes) as part of the EMS data pipeline. The BPI-F3 has a SpacemiT K1 octa-core RISC-V CPU @ up to 1.6 GHz with 2.0 TOPS NPU and LPDDR4 RAM (options for 2/4/8/16 GB) [4].

Industrial-grade alternatives to Raspberry Pi. For EMS gateways with industrial requirements (long-term availability, temperature ranges, I/O) several platforms go beyond hobbyist SBCs:

i. *Pi.MX8* (open-hardware CM4-compatible SoM) brings NXP i.MX 8M Plus (quad Cortex-A53 up to 1.8 GHz plus Cortex-M7 @ 800 MHz and 2.3 TOPS NPU, typically configured with 1–4 GB LPDDR4) into the Raspberry Pi CM4 form factor, enabling reuse of established carrier boards while offering industrial longevity and on-device AI for forecasting/optimization [14, 21].

ii. *Intel NUC (with Enapter Gateway)* provides an industrial-grade EMS solution when paired with Enapter Gateway software. Recommended configurations include Intel® NUC 10th generation Core™ i3-10110U @ 2.1 GHz or i5-10210U @ 1.6 – 4.2 GHz with dual-channel DDR4 RAM (commonly 8 GB, up to 64 GB). This platform supports real-time device communication, monitoring, and control via protocols such as MQTT, OPC UA, Siemens S7, Modbus, and REST APIs [10].

Other open-source EMS stacks and their hardware ecosystems. Beyond OpenEMS, other common open hardware/-software stacks are:

i. *OpenRemote* provides an open-source IoT/EMS stack oriented to buildings and microgrids, integrating heterogeneous assets, forecasts, and optimization; it typically targets x86/ARM/RISC-V Linux SBCs [15, 16].

ii. *OpenEnergyMonitor* (OEM) combines open hardware (emonTx/emonPi) and software for home energy monitoring, often leveraging Raspberry Pi and Arduino-class nodes; while more monitoring-focused, OEM illustrates a mature open HW/SW ecosystem widely used in residential contexts [18, 22].

Energy-related open-source software in general is listed at [11].

Takeaways. EMS hardware spans both low-cost SBCs (Raspberry Pi/Banana Pi) for prototyping or small sites and industrial SBCs/SoMs (e.g., NXP i.MX8 family) for production gateways with long-term support. The processors used are usually ARM application-class chips.

3. Get a rich Linux RISC-V environment quickly

On a host laptop running Debian Linux, we installed the packages `qemu-system-misc`, `u-boot`, and `opensbi` to have the QEMU emulator [23] running as initial development environment. For the target running on QEMU, we used the Debian image builder from [3] to obtain a QEMU Debian image, using the Debian testing version, because, as of now, OpenJDK RISC-V ports are not yet available in Debian stable.

OpenJDK RISC-V ports are available since version 22, to ensure stability, we installed the version OpenJDK 23 (openjdk-23-jre-headless). In the QEMU configuration, we explicitly specified 48-bit addresses (“`-cpu rv64,sv57=off`”), to ensure 48-bit address handling, as OpenJDK-23 did not yet support 57-bit addresses.

4. OpenEMS energy management application

We use a configuration for the OpenEMS open-source energy management system (OpenEMS [19]), built on top of the Java OSGi framework [20]. We targeted a simulated home energy management system setup, provided by OpenEMS as an introductory example, consisting of a photovoltaic producer, electricity consumers and a battery, where the energy management system decides on whether to feed the photovoltaic yield to the electricity grid or to charge a battery instead.

We took OpenEMS from GitHub and found out that it mostly runs on RISC-V except for the use of an OSGi package, which utilizes a platform-dependent JNA (Java Native Applications), which we were able to disable. We used the tutorial to create a run-time image on the host and deployed it on the target.

5. Running on RISC-V CVA6 (Genesys2 FPGA)

We generated a bitstream and memory configuration file, then loaded the software to the SD card with the Debian environment built in Sections 3 and 4 and changeroot into it. For a full tutorial on how to reproduce these steps, see [6].

An overview of the output shown in Figure 1, this a shortened version of the screen output captured in `screenlog.0`, with omissions marked by “`[...]`”: the first five lines are from the FPGA, then follows OpenSBI output, e.g., naming the platform as “`ARIANE RISC-V`” which refers to CVA6, Linux “`Starting kernel` ...”, then we see our chroot and starting the demo. The output starting with “`org.ops4j`” is from the OSGi framework, in the lines with the timestamp “`1970-01-01T03:03:11,678`” we see output from the OpenEMS demonstration setup, that is, a home energy management system with a simulated electricity consumption (in this initial line) of 919 W, of which 688 W are provided from the grid, and are 231 W provided from a battery with 50% load state. The next two lines shown simulate fluctuations in consumption and battery control (e.g., the battery taking a higher part in providing the electrical energy), of course after the system has started, many more lines follow in the control loop.

```
init SPI
status: 0x0000000000000025
status: 0x0000000000000025
SPI initialized!
[...]
OpenSBI v0.9
[...]
Platform Name : ARIANE RISC-V
Platform Features : medeleg
Platform HART Count : 1
Platform IPI Device : aclint-mswi
Platform Timer Device : aclint-mtimer @ 1000000Hz
Platform Console Device : uart8250
[...]
Starting kernel ...
[ 0.000000] Linux version 5.10.7 (hbl@hbl-lap-14) (riscv64-
buildroot-linux-gnu-gcc.br_real (Buildroot 2021.08) 10.3 .0, GNU
Id (GNU Binutils) 2.36.1) #3 SNP Tue Jan 16 12:15:14 CET 2024
[...]
# chroot /mnt
# java -jar demo.jar
[...]
org.ops4j.pax.logging.pax-logging-api
[org.ops4j.pax.logging.internal.Activator] INFO : Enabling Java
Util Logging API support.
[...]
1970-01-01T03:03:11,678 [_cycle ] INFO
[ebuglog.ControllerDebugLogImpl] [ctrlDebugLog0] _sum[State:Fault
Ess SoC:50 %|L:231 W Grid:688 W Consumption:919 W] ess0[SoC:50
%|L:231 WiAllowed:-10000;10000 W]
```

Figure 1. OpenEMS boot on RISC-V CVA6 running on Genesys2 FPGA (console output from board, OpenSBI, Linux and OpenEMS)

6. Experimental setup with OpenEMS

To complement the evaluation of OpenEMS on RISC-V hardware, we conducted an additional study examining how the framework processes real, device-level measurements from a residential photovoltaic installation. While earlier assessments relied on synthetic traces generated within the OpenEMS simulator, this experiment used timestamped CSV logs from a commercial household PV system investigated in the ISOLDE project, containing realistic grid-exchange and photovoltaic production values recorded during typical summer operation. Following the standard OpenEMS runtime and configuration workflow, these datasets were registered as CSV-based data sources and mapped to simulated PV inverter and grid-meter components with only minor scaling adjustments, enabling OpenEMS to infer household consumption directly from inverter telemetry. The runtime operated with stable one-second cycles, and the OpenEMS UI rendered synchronized energy flows with sub-second latency dominated by browser and network overhead. Together with the RISC-V deployment results, this demonstrates that open instruction set platforms are ready to run complex workloads like OpenEMS on open, vendor-independent hardware, forming a sustainable, device-agnostic alternative to established platforms for decentralized residential energy management.

7. Putting a hypervisor beneath

Having already validated the integration of the applications with Linux, in this step, we add a hypervisor beneath, in order to show running embedded Linux with application load in a secure environment. We use SYSGO's CODEO GUI [27], which allows for graphical configuration of embedded systems, including board parameters, a hypervisor, and optionally embedded Linux guests, where we build an integration project with the PikeOS hypervisor. As part of this integration project, we also equipped it with an ELinOS embedded Linux. For simplicity, akin to a the chroot approach we took on CVA6 before, we manually replaced the embedded Linux root file system (`rootfs`) by a `squashfs` image generated from the aforementioned Debian Linux RISC-V environment, augmented by an init process and support for mounting an overlay file system to run OpenEMS on openjdk.

As the Java virtual machine on-the-fly generates machine instructions from bytecode, one feedback that we got from this exercise was the necessity to enable instruction cache operations.

8. StreamPipes

Another popular tool commonly used for analyzing industrial IoT data, such as energy data, is Apache StreamPipes [1], an Apache Software Foundation top-level project. StreamPipes focuses on an end-to-end toolbox helping non-technical users to analyze continuous data streams from machines, sensors, and other devices (Figures 2). The toolbox comprises both user-oriented (e.g., user interfaces for data connectivity and pipeline-based algorithm orchestration) and developer-oriented tools (e.g., Python client libraries or an SDK for protocol extensions). With a distributed architecture consisting of various microservices, StreamPipes is a suitable tool for managing IoT data on a larger scale. In many scenarios, so-called *Extension Services* can be deployed at edge nodes (e.g., industrial PCs). These services are responsible for data collection and preprocessing from devices like PLCs or energy management controllers. Extension services publish data to the StreamPipes core using messaging systems such as MQTT [5].

Users can remotely deploy connectors to edge devices by using the adapter library available in the StreamPipes user interface. Based on registered *tags*, the system identifies suitable edge nodes and remotely starts an adapter by sending an invocation message. In this scenario, we experimented with a distributed setup where a single extension service is running on a RISC-V-based Banana Pi F3 device. The device is connected to an energy management controller (in our case, a Consolinno Leaflet) which is running an OPC-UA server. To achieve this, we optimised the extension service in a number of ways. Firstly, in order to reduce the memory footprint, we created a lightweight service that only supports one messaging protocol (NATS) and a subset of the available StreamPipes protocol implementations. This also helps to reduce startup time. Secondly, having experimented with building a RISC-V-compatible container image, we encountered a startup problem related to denied syscalls (`RISCV_FLUSH_ICACHE`). Instead of a container-based deployment approach, we ran a plain Java build using the Eclipse Temurin JDK for RISC-V.

From our experiments, we conclude that porting the StreamPipes extension service was rather straightforward and mainly required modifications in terms of improving resource efficiency on the application level. A minor drawback is the startup time, which takes roughly 30 seconds on the tested Banana Pi device - however, this does not affect real-world applicability since in most cases the service is started once and then continuously fetches data from downstream devices, which has proven to be of adequate performance. In general, we believe that once the average hardware costs of RISC-V-based, resource-efficient edge devices are more closely aligned with other architectures, running StreamPipes extension services on RISC-V are a good way to establish large-scale, distributed deployments in edge-cloud IoT scenarios where a central layer is inappropriate (e.g., due to firewall restrictions).

To further evaluate the suitability of RISC-V for IoT scenarios, we used the Rocket Chip Generator [2] to design a tailored RISC-V-based SoC with an integrated ethernet controller. This design was deployed on a Xilinx KR260 board in order to create a lightway gateway to augment the energy data with additional sensor measurements. To ensure the efficient operation of the resource-constrained platform, we built the software stack on Zephyr RTOS [29], leveraging its built-in MQTT client library as foundation for a custom MQTT relay server.

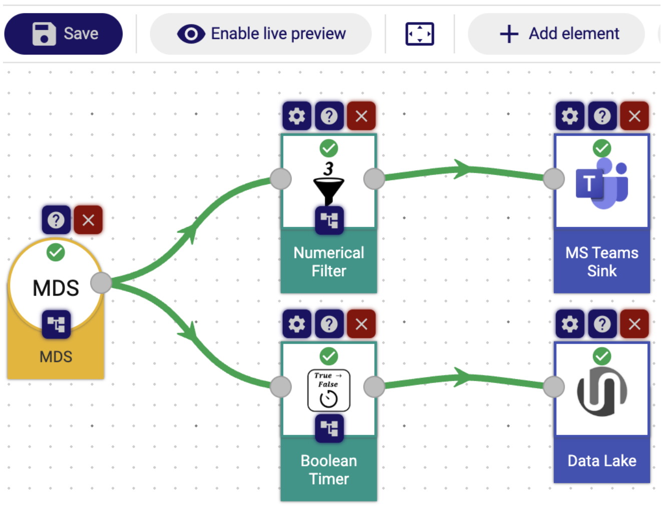

Figure 2. Apache StreamPipes Pipeline Editor

Thus, this setup allowed data to be collected from external networked sensors acting as local MQTT clients, as well as from environmental sensors that were connected directly to the board. All sensor data was aggregated and forwarded to the StreamPipes server via MQTT for further analysis. The system performed well in practice, demonstrating the adaptability of RISC-V even for small, resource-limited IoT platforms.

9. Discussion and ongoing/further work

We have described running modern Java applications on RISC-V, thus validating RISC-V ecosystem maturity, such as (1) portability of OpenEMS, (2) ability to run complex OSGi Java applications, (3) maturity of the open hardware CVA6 platform and QEMU, (4) maturity of system software stacks with the Debian and ELinOS Linux distributions and the PikeOS hypervisor to support this setup, (5) running an Apache StreamPipes extension service. Running on FPGA of course is much slower than on real hardware, but it shows that all functionality is available. We have shown our incremental approach, the intent is experience-sharing, in the spirit of other RISC-V learning resources [25].

Specifically on the OpenEMS integration side, we had started with a stack of CVA6 on FPGA, Debian and OpenEMS (Section 5), then we have run a stack of RISC-V PolarFire PikeOS, ELinOS kernel, Debian file system, OpenJDK and OpenEMS (Section 7). Next, we have ported PikeOS to the dual-core CVA6 platform (Culsans [9]) and we are working on replicating the ELinOS/Debian/OpenJDK/OpenEMS setup here. Also, we intend to test certain optional CVA6 extensions, such as caches, with our payloads and possibly use our experience to validate our own developments on a tracing port (Trace Ingress Port [26]) on the software side.

Although not within our scope, it is conceivable that our setups could be re-used for some sort of more ready-made test bench. In parallel, a secure and modular pipeline for data acquisition, cryptographic processing, and visualization of energy management data is implemented for RISC-V based IoT devices. The data flow starts at the microcontroller, which acquires sensor or grid data and applies AES-256 encryption to ensure confidentiality during transmission to OpenEMS.

10. Acknowledgment

This work was partially supported by the European Union projects TRISTAN (RIA Chips-JU, GA 101095947) as well as ISOLDE (RIA Chips-JU, GA 101112274). We also thank our ERTS shepherd for feedback on a previous draft version and initial OpenEMS hands-on training from the Göttingen September 2023 OpenEMS hackathon participants.

References

[1] Apache Software Foundation. Apache StreamPipes, 2025. URL https://streampipes.apache.org/.

[2] Krste Asanović et al. The rocket chip generator. Technical Report UCB/EECS-2016-17, Univ of California, 2016.

[3] Colin Atkinson. Getting started with RISC-V in QEMU, January 2021. URL https://colatkinson.site/linux/riscv/2021/01/27/riscv-qemu/.

[4] Banana Pi Project. Banana pi bpi-f3 documentation, 2026. URL https://docs.banana-pi.org/en/BPI-F3/BananaPi_BPI-F3. Accessed: 2026-01-08.

[5] A. Banks and R. Gupta. MQTT version 3.1.1. Standard, OASIS, 2014. URL https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html.

[6] H. Blasum, D. Sheladiya, J. Reinhard, F. Krebs, D. Engraf, S. Ardaya-Lieb, F. Haxel, D. Riemer, G. Suciu, M. Sachian, and R. Florescu. Porting OpenEMS and Apache StreamPipes energy management software to freshly developed RISC-V system software stacks: Detailed OpenEMS setup on host and targets, January 2026. URL https://doi.org/10.5281/zenodo.18164662.

[7] Catalin Bogdan Ciobanu et al. The TRISTAN and ISOLDE RISC-V projects, 2025. SAMOS 2025 conference proceedings, accepted.

[8] OpenEMS Community. Optimal hardware requirements for openems (forum thread). https://community.openems.io/t/optimal-hardware-requirements-for-openems/2495, 2024.

[9] Culsans. Culsans - tightly-coupled cache coherence unit using the ACE protocol, 2025. URL https://github.com/pulp-platform/culsans.

[10] Enapter AG. Enapter gateway — industrial-grade energy controller. https://fw.enapter.com/, 2025.

[11] Linux Foundation Energy. Energy-related open-source project ecosystem, 2026. URL https://landscape.lfenergy.org/?group=energy-related-open-source-project-ecosystem.

[12] OpenEMS Association e.V. Openems - open source energy management system (github). https://github.com/OpenEMS/openems, 2025.

[13] OpenEMS Association e.V. Introduction :: Open energy management system, 2025. URL https://openems.github.io/openems.io/openems/latest/introduction.html.

[14] Hackster.io. Pi.mx8: Cm4-compatible i.mx 8m plus som with npu. https://www.hackster.io/news/ov-tech-s-pi-mx8-offers-a-raspberry-pi-compute-module-4-alternative-with-on-board-ai-acceleration-4a377e1fdb7b, 2023.

[15] OpenRemote Inc. Openremote platform overview. https://openremote.io/, 2025.

[16] OpenRemote Inc. Open source iot platform for energy management. https://openremote.io/energy-management-open-source/, 2025.

[17] ISOLDE project. ISOLDE home page, 2025. URL https://www.isolde-project.eu.

[18] MDCPlus. Top free & open-source energy & sustainability monitoring systems. https://mdcplus.fi/blog/top-free-energy-monitoring-systems-sustainability/, 2025.

[19] OpenEMS Association e.V. OpenEMS - Open Source Energy Management System, 2016. URL https://github.com/OpenEMS/openems.

[20] OSGi Working Group. OSGi: The Dynamic Module System for Java, 2025. https://www.osgi.org/.

[21] OVTech. Pi.mx8m plus som (open hardware share). https://www.pcbway.com/project/shareproject/Pi_MX8M_Plus_SoM_e3fe3cd9.html, 2022.

[22] OpenEnergyMonitor Project. Openenergymonitor overview. https://openenergymonitor.org/homepage/about, 2025.

[23] QEMU. QEMU: A generic and open source machine emulator and virtualizer, 2025. URL https://www.qemu.org/.

[24] Raspberry Pi Ltd. Raspberry pi 5 product brief, 2023. URL https://pip-assets.raspberrypi.com/categories/892-raspberry-pi-5/documents/RP-008348-DS-4-raspberry-pi-5-product-brief.pdf. Accessed: 2026-01-08.

[25] RISC-V International. riscv/learn: Tracking RISC-V Actions on Education, Training, Courses, Monitorships, etc., 2024. URL https://github.com/riscv/learn.

[26] Darshak Sheladiya. Add support for Trace Ingress Port (TIP) on CVA6 V5.1.0, 2024. URL https://github.com/openhwgroup/cva6/pull/2601.

[27] SYSGO GmbH. CODEO Development Environment, 2025. URL https://www.sysgo.com/codeo.

[28] TRISTAN project. Expand, mature and industrialise the European RISC-V ecosystem, 2025. URL https://tristan-project.eu/.

[29] Zephyr Project. Zephyr RTOS, 2025. URL https://zephyrproject.org/.