Enabling RISC-V CI in Open-Source Projects: Challenges and Solutions

Marek Pikuła[1]

[1](Samsung R&D Institute Poland)

Abstract

The adoption of RISC-V as a viable architecture for open-source software development is gaining traction. However, a major challenge remains: ensuring continuous integration (CI) support for RISC-V in upstream projects. At Samsung R&D Institute Poland, we addressed this issue by enabling RISC-V CI for several Freedesktop.org (FDO) projects, including Pixman and GStreamer Orc, and we are currently extending our work to the Opus codec. This work presents our approach to enabling RISC-V CI in FDO projects, addressing the challenges of testing architecture-specific optimizations without native hardware support. We detail our implementation of Docker-based GitLab runners with QEMU emulation, enabling automated multi-architecture testing while minimizing infrastructure overhead. Our work not only enhances software quality by enabling automated testing for RISC-V but also provides a framework for future contributions to seamlessly integrate RISC-V into open-source CI ecosystems.

Introduction

The RISC-V ecosystem has seen rapid expansion, yet upstream open-source projects often lack Continuous Integration (CI) support for this architecture. Many projects primarily test on x86 and ARM due to limited infrastructure availability. This lack of CI resources prevents maintainers from validating RISC-V-specific optimizations, creating a barrier to broader adoption.

This limitation became evident when integrating RISC-V Vector extension (RVV) support into the Pixman library. Without access to RISC-V hardware, maintainers faced difficulties in validating the implementation. To address this, we collaborated with Freedesktop.org (FDO) CI maintainers to develop an efficient testing workflow that verifies architecture-specific backends without overwhelming limited CI resources. We are also in the process of extending our approach to additional projects, including GStreamer Orc and Opus.

Overview of Targeted Libraries

Pixman

Pixman[1] is a low-level pixel manipulation library used by various graphics stacks, including the X server and Cairo. It features architecture-specific SIMD backends for optimized performance on different platforms such as x86 (MMX/SSE2, SSSE3), ARM (NEON), MIPS (DSPr2), PowerPC (AltiVec/VMX), and LoongArch (MMI). SIMD (Single Instruction, Multiple Data) is a parallel computing technique that accelerates processing by performing the same operation on multiple data points simultaneously.

GStreamer ORC

GStreamer ORC (The OIL Runtime Compiler)[2] is a library that dynamically generates vectorized code from ORC bytecode, enhancing media processing performance. Similar to Pixman, it contains multiple architecture-specific backends.

Opus

Opus[3] is a widely adopted audio codec designed for high-quality, low-latency voice and music streaming. Its architecture-specific optimizations, providing benefits in the processing speed and energy efficiency, make it a prime candidate for RISC-V enablement, particularly with RVV support.

Creating the CI Workflow

FDO's GitLab CI infrastructure is limited to x86 and ARM, as it relies on donated computing resources. To enable RISC-V testing without adding significant maintenance overhead, we implemented a solution using Docker-based GitLab runners with QEMU user-mode emulation[4].

Our primary objectives were to:

- Provide CI coverage for all supported architectures (including basic support for Windows using Wine).
- Use native ARM runners where applicable.
- Build the libraries with both GNU and LLVM toolchains.
- Execute tests for all SIMD backends.
- Generate a consolidated coverage report.

Docker Image Preparation

We developed reusable GitLab CI templates to build multi-architecture Docker images, leveraging extensive Debian's cross-architecture support and pre-built base images.

To enable comprehensive testing, our images included necessary toolchains (GNU, LLVM), required library dependencies, and tooling for analyzing test results.

Using a multi-stage, composable Docker image approach, we ensured flexibility—allowing projects to either use pre-built CI images (GNU, LLVM, Meson stack) or extend them for project-specific needs (e.g., by installing required system dependencies).

Whenever possible, we prioritized native execution to simplify debugging and enable coverage analysis. However, for certain architectures (e.g., big-endian PowerPC 32-bit), cross-compilation was required, in which case we focused solely on correctness verification.

Build Stage

Initially, Pixman's CI workflow only ran tests with the GNU toolchain, leading to overlooked issues in LLVM builds. Architecture-specific code (e.g., with compiler intrinsics) can behave differently between compilers, necessitating dual-toolchain testing. Including both GNU and LLVM in our workflow uncovered several previously unreported LLVM-related bugs, improving the library's overall reliability.

Test Stage

The CI workflow executes tests across all supported architecture-specific backends. For instance, x86 builds reuse binaries across MMX, SSE2, and SSSE3 SIMD implementations allowing for running the tests in parallel.

Similarly, for RVV, we tested multiple vector register lengths (VLENs) to identify configuration-specific issues. Since development hardware we used supported a VLEN of 256, automated testing of various configurations ensured early discovery of potential issues for future RISC-V targets.

Summary Stage

Native execution simplified coverage analysis, but each test run generated separate reports. Manually reviewing over 40 reports was impractical, so we used *gcovr*[5] to merge them into a unified summary, producing both human-readable and machine-consumable reports for GitLab's CI system[6].

Impact and Future Work

Our methodology has already uncovered early-stage bugs in Pixman's RVV implementation by testing across various VLEN configurations. The same approach will be extended to GStreamer Orc to enhance RISC-V support in other media processing workloads.

For Opus, automated RISC-V testing significantly accelerates development, reducing reliance on manual hardware validation. Our work establishes a scalable model for integrating RISC-V into CI workflows.

Looking ahead, we plan to expand our methodology to additional projects and collaborate with open-source communities to further enhance RISC-V CI capabilities.

It's worth to mention the Cloud-V CI/CD system[7], which provides real, hardware RISC-V targets for use in CI/CD workflows. This looks like an interesting option for the future, in case a project requires, e.g., performance measurement in CI, which cannot be provided in QEMU environment.

Conclusion

By enabling RISC-V CI in upstream open-source projects, we have addressed one of the key barriers to RISC-V adoption. Our approach is generic enough that it can be reused in other Freedesktop.org projects and in external GitLab instances used by other open source projects. This work ensures that maintainers can validate RISC-V contributions without requiring dedicated hardware, making it easier for developers to contribute RISC-V optimizations. This success story demonstrates how strategic CI integration can accelerate the growth of RISC-V within the open-source ecosystem.

References

[1] Freedesktop.org. *Pixman*. URL: https://pixman.org/.

[2] Freedesktop.org. *GStreamer ORC*. URL: https://gitlab.freedesktop.org/gstreamer/orc.

[3] Jean-Marc Valin, Koen Vos, and Timothy B. Terriberry. *Definition of the Opus Audio Codec*. RFC 6716. Sept. 2012. DOI: 10.17487/RFC6716. URL: https://www.rfc-editor.org/info/rfc6716.

[4] Docker Inc. *Docker: Multi-platform builds*. URL: https://docs.docker.com/build/building/multi-platform/.

[5] Lukas Atkinson and Michael Förderer. *gcovr*. URL: https://www.gcovr.com/en/stable/index.html.

[6] GitLab Inc. *GitLab: Test coverage visualization*. URL: https://docs.gitlab.com/ee/ci/testing/test_coverage_visualization/.

[7] 10xEngineers. *Cloud-V: Continuous Deployment and Continuous Integration (CI/CD)*. URL: https://cloud-v.co/ci-cd.