RISC-V VP++: Next Generation Open-Source Virtual Prototype

Manfred Schlägl Christoph Hazott Daniel Große
Institute for Complex Systems, Johannes Kepler University Linz, Austria
{manfred.schlaegl,christoph.hazott,daniel.grosse}@jku.at

***Abstract*—In this paper, we present the heavily extended open-source *RISC-V VP++*, which combines several VP-based projects into a powerful new tool to support early design space exploration, evaluation, verification and validation of RISC-V based systems at the system level. The paper briefly describes three projects that started as independent works and highlights the associated synergies and great new potentials that arise from integrating them into a single solution.**

I. Introduction

*Virtual Prototype*s (VPs) are high-level, executable models of the entire *Hardware* (HW) platforms which can run unmodified production *Software* (SW) [1], [2]. VPs allow early design space exploration, and system evaluation and validation *before* a physical HW is built or even designed. Today, VPs are predominantly implemented in SystemC, a standardized class library for C++ (IEEE 1666, [3]) [4]. Based on the abstraction of communication details by leveraging *Transaction Level Modeling* (TLM) [5], the simulation is orders of magnitude faster in comparison to RTL [1].

A central element of a HW platform is the processor. In the recent years, the open and royalty-free *Instruction Set Architecture* (ISA) RISC-V [6], [7], which features an extremely modular design, gained enormous momentum in academia and industry. In RISC-V, modularity is achieved by a variety of standard extensions that can be added to a very small base ISAs according to application specific requirements. One of these extensions is the *RISC-V "V" Vector Extension* (RVV) [8], which adds *Single Instruction, Multiple Data* (SIMD) functionality to the RISC-V architecture. In SIMD, operations are applied not only to single data elements but on whole *vectors* of elements simultaneously. SIMD takes advantage of *Data-Level Parallelism* (DLP) and can significantly improve data throughput and thus the performance of parallelizable algorithms such as those used in machine learning and multimedia applications [9]. An overview of RVV is depicted in Fig. 1. It is worth noting that, unlike in classic SIMD (e.g. ARM Neon), the length of the vector registers (`VLEN`) is not fixed in the ISA, but can be chosen by the designer. SW can determine `VLEN` at runtime and automatically adapt to the capabilities of a specific HW.

The open-source *RISC-V VP++* considered in this paper was introduced in [10] and is available on GitHub[1]. The VP comes with *Instruction Set Simulator*s (ISSs) for RISC-V in 32 (*RV32*) and 64 bit (*RV64*) configurations. A TLM bus connects the ISSs, memory and the peripherals. A *Core Local Interruptor* (CLINT) and *Platform-Level Interrupt Controller* (PLIC) provide support for timer and interrupts. The VP includes different configurations that model small microcontroller-based, up to larger application processor-based platforms. Small configurations lack support for *Virtual Memory Management* (VMM) and are typically used to run bare-metal SW or small *Operating System*s (OSs) systems. Larger configurations come with VMM and can run general purpose OSs, like Linux.

[1]https://github.com/ics-jku/riscv-vp-plusplus

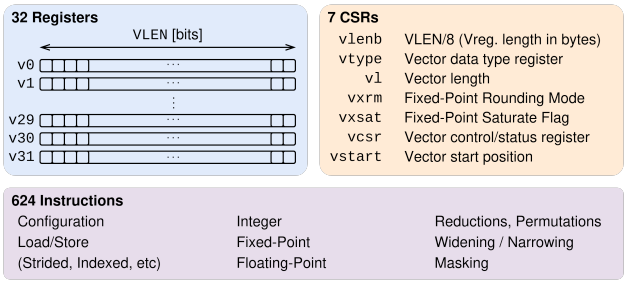

Fig. 1: Overview of the *RISC-V "V" Vector Extension* (RVV)

II. Extensions of *RISC-V VP++*

A. RISC-V "V" Vector Extension

The extension of *RISC-V VP++* with the ratified RVV version 1.0 [8] is presented in [11]. The paper describes the integration of the 32 vector registers, 7 RVV *Control and Status Register*s (CSRs) and 624 RVV instructions (as shown in Fig. 1) in the RV32 and RV64 ISSs of *RISC-V VP++*. The RVV extended VP is verified via comparing execution traces of randomly generated instruction sequences against a golden model. In total, a functional coverage of 81.44% (*riscvOVPsim* basic coverage for RVV) is achieved. The paper also demonstrates the value of the RVV extended VP for system-level evaluation with a case study. However, the value goes beyond the presented case study: The integration of RVV in the RV32 and RV64 ISSs of *RISC-V VP++* enables support for RVV in all configurations contained in *RISC-V VP++*. This opens up a wide range of possibilities for experimentation (e.g. system, HW, system SW, application SW) with platforms at different scales (from microcontroller- up to application processor-based).

B. GD32V

GD32V is a RISC-V based microcontroller family from GigaDevices. *RISC-V VP++* includes a model and configuration of one derivate of this family, the GD32VF103VBT6 *System-On-Chip* (SoC). The SoC comes with large set of peripherals which include, for example, general purpose IO (+interrupts), *Serial Peripheral Interface* (SPI) and the external parallel bus. To make the GD32V configuration practically

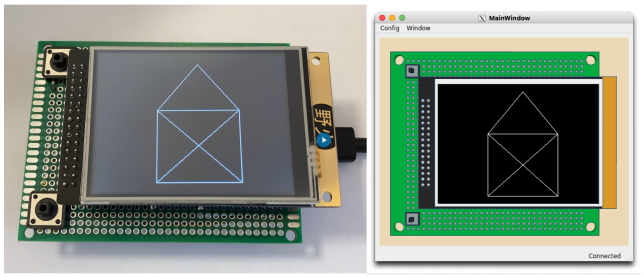

Fig. 2: Real and Virtual Environments

useful, the model also includes external peripherals, namely a display (connected via parallel bus) and a touchscreen (connected via SPI). To interact with the platform, *RISC-V VP++* includes a GUI that can be connected to the running model. Fig. 2 shows the execution of the exact same program on the real HW (left) and the VP+GUI (right).

The value of *RISC-V VP++* with GD32V was demonstrated in [10] where an approach for virtual verification of embedded graphics libraries is introduced leveraging a VP and metamorphic testing leading to uncover 15 previously unknown bugs in a widely used library. The combination with RVV, presented in Sec. II-A, opens up an even wider range of possibilities: RVV is not seen in small scale systems today. *RISC-V VP++* with GD32V can, for example be used as a powerful tool for the evaluation of RVV for small scale systems.

C. GUI-VP Kit *and* RISC-V VP++

GUI-VP Kit and *GUI-VP* were introduced in [12] and provide a full RISC-V development and simulation environment for interactive graphical Linux applications. *GUI-VP* is a greatly extended and improved *RISC-V VP*, that comes with new TLM peripherals providing graphical output, and mouse and keyboard input by adopting *Virtual Network Computing* (VNC). Additionally, the VP introduces a CLINT implementation that ensures real-time behavior for interaction. *GUI-VP Kit* includes *GUI-VP*, comes with drivers for the new peripherals and provides everything necessary to build a runnable Linux environment. In summary, *GUI-VP Kit* allows for execution of complex Linux graphics frameworks and applications, as demonstrated in Fig. 3 and Fig. 4. Fig. 3 presents a running X.Org desktop environment with applications. Fig. 4 shows a Linux port of a classic 3D game (*PrBoom*) reaching up to 8.8 frames per second.

All modifications of *GUI-VP* are now fully integrated and maintained in *RISC-V VP++*, and *GUI-VP Kit* is migrated to use *RISC-V VP++*. With this, users of *GUI-VP Kit* can now benefit from all other extensions made to *RISC-V VP++*. Examples of this are the optimization of the boot time and support for persistent storage through the use of memory mapped file system images. However, the synergy with the highest potential is the combination with RVV from Sec. II-A. By migrating *GUI-VP Kit* to *GCC-13* and *Linux-6.6*, we have enabled support for RVV. Consequently, *GUI-VP Kit* is now also a quick to create and easy to use Linux + RVV experimentation environment. This allows not only the evaluation of DLP algorithms and RVV HW implementations, but also, for example, the evaluation of system SW, such as the handling

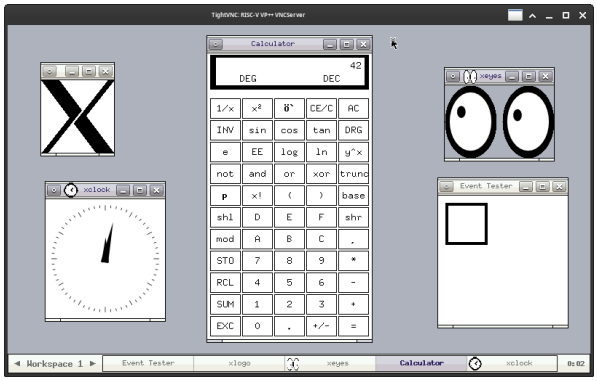

Fig. 3: X.Org desktop and applications running on *RISC-V VP++*

Fig. 4: PrBoom running on *RISC-V VP++*

of RVV in the Linux kernel. First successful experiments were already performed with our RVVRADAR [13] framework.

ACKNOWLEDGMENTS

This work has partially been supported by the LIT Secure and Correct Systems Lab funded by the State of Upper Austria.

REFERENCES

[1] T. De Schutter, *Better Software. Faster!: Best Practices in Virtual Prototyping*. Synopsys Press, March 2014.
[2] R. Leupers, G. Martin, R. Plyaskin, A. Herkersdorf, F. Schirrmeister, T. Kogel, and M. Vaupel, "Virtual platforms: Breaking new grounds," in *DATE*, 2012, pp. 685–690.
[3] "IEEE standard for standard SystemC language reference manual." [Online]. Available: https://doi.org/10.1109/ieeestd.2012.6134619
[4] V. Herdt, D. Große, and R. Drechsler, *Enhanced Virtual Prototyping: Featuring RISC-V Case Studies*. Springer, 2020.
[5] *OSCI TLM-2.0 Language Reference Manual*, OSCI, 2009. [Online]. Available: https://www.accellera.org/images/downloads/standards/systemc/TLM_2_0_LRM.pdf
[6] A. Waterman and K. Asanović, *The RISC-V Instruction Set Manual; Volume I: Unprivileged ISA*, SiFive Inc. and UC Berkeley, 2019.
[7] ——, *The RISC-V Instruction Set Manual; Volume II: Privileged Architecture*, SiFive Inc. and UC Berkeley, 2019.
[8] "RISC-V V vector extension," https://github.com/riscv/riscv-v-spec, 2022.
[9] M. Flynn, "Very high-speed computing systems," *IEEE*, vol. 54, no. 12, pp. 1901–1909, 1966.
[10] C. Hazott, F. Stögmüller, and D. Große, "Verifying embedded graphics libraries leveraging virtual prototypes and metamorphic testing," in *ASP-DAC*, 2024.
[11] M. Schlägl, M. Stockinger, and D. Große, "A RISC-V "V" VP: Unlocking vector processing for evaluation at the system level," in *DATE*, 2024.
[12] M. Schlägl and D. Große, "GUI-VP Kit: A RISC-V VP meets Linux graphics - enabling interactive graphical application development," in *GLSVLSI*, 2023, pp. 599–605.
[13] L. Klemmer, M. Schlägl, and D. Große, "RVVRadar: a framework for supporting the programmer in vectorization for RISC-V," in *GLSVLSI*, 2022, pp. 183–187.