2022 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE
MUNICH, GERMANY
DECEMBER 6 - 7, 2022

SIM-V

Fast, Parallel RISC-V Simulation for Rapid Software Verification

Lukas Jünger, MachineWare GmbH, Aachen, Germany (lukas@mwa.re)
Jan Henrik Weinstock, MachineWare GmbH, Aachen, Germany (jan@mwa.re)
Rainer Leupers, RWTH Aachen University, Aachen, Germany (leupers@ice.rwth-aachen.de)

***Abstract* — Over the last decades HW/SW software systems have become increasingly complex. Today, even small embedded systems are made up of different IP blocks from many vendors and execute complex software stacks consisting of millions of lines of code. This complexity ultimately leads to an increased risk of software bugs and security vulnerabilities that compromise the system's security and safety. To detect these issues early in the design cycle, full system simulators based on the SystemC TLM-2.0 standard, so called Virtual Platforms (VPs), are the goto tool. Unfortunately, high target system complexity usually leads to reduced VP simulation performance. Contemporary VPs often fail to deliver the performance required for executing realistic workloads in a reasonable time frame. During target software development this is especially problematic as live software development requires close to real time feedback from the simulation. In Continuous Integration (CI) scenarios slow VPs prohibit testing of every commit and often only a daily CI run is possible, making it hard to pinpoint which exact change introduced faulty behavior. To alleviate these problems we introduce SIM-V. SIM-V is a SystemC TLM-2.0 based RISC-V simulator targeted for early software development and verification. Its high-performance RISC-V processor models, based on our custom JIT engine FTL, enable the thorough verification of large target software stacks and by supporting several open integration layers, they can easily be integrated into existing VPs. In the presented case study, we show that SIM-V outperforms QEMU by a factor of 2x in both sequential and parallel execution, even though SIM-V has to carry the SystemC kernel and annotates timing information. Through integration with our open-source SystemC TLM-2.0 productivity library VCML, SIM-V offers Python scripting capabilities that enable deep introspection and instrumentation for seamless integration into complex CI scenarios.**

Keywords—RISC-V, ESL, SystemC TLM-2.0, Instruction Set Simulation, Continuous Integration

I. Introduction

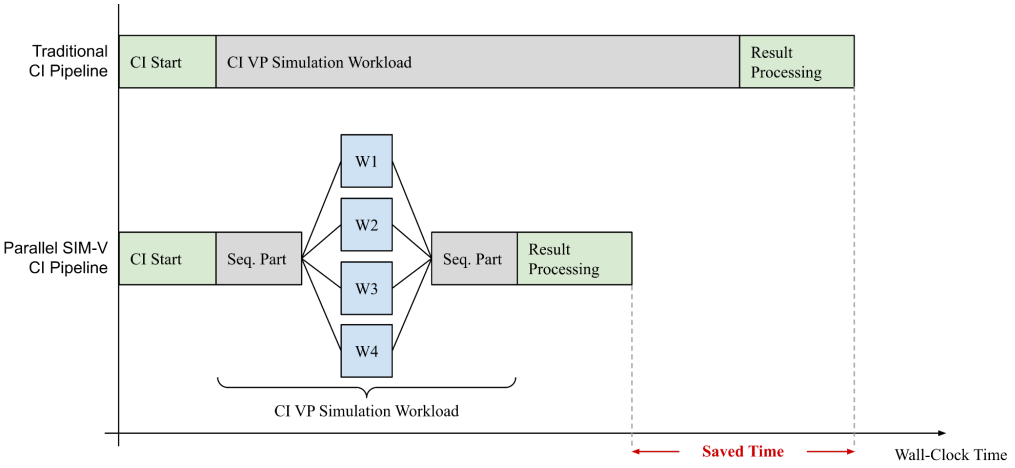

Figure 1. CI pipeline speed-up by using parallel SIM-V.

Full system simulators, so called Virtual Platforms (VPs), have become the de facto standard tool for early target software development and verification before physical hardware prototypes become available. The need to deploy VPs early in the design cycle has been increasing steadily over the last decades, as HW/SW system complexity keeps rising. Keeping up with today's tight product schedules without early simulator deployment seems next to impossible.

However, as target system complexity increases so does the complexity of the VP. This leads to simulation performance issues, as traditional SystemC TLM-2.0 simulation is sequential and adding more components to the simulation slows execution of the entire simulator. Especially, when used for software development VP performance is critical, as close to real-time speed is required for a good VP user experience. Here, the performance of the Instruction Set Simulator (ISS) plays a key role, as the execution of the target software drives the entire simulation. Besides deploying a high-speed ISS, parallelization of the VPs processor core models on multiple host CPU cores is required to increase performance.

Besides early software development, Continuous Integration (CI) is an important VP use case. Here, VPs can show off their potential as they can be easily scaled given that enough compute resources are provided, while physical hardware prototypes are usually only available in small quantities. Modern embedded systems execute huge software stacks consisting of millions of lines of code. Executing comprehensive test suites for every single commit is paramount for ensuring the early detection of bugs that compromise the system's security and safety. Again the performance of the ISS plays a critical role in achieving high simulation performance, which is required to execute all tests in a reasonable amount of time. Besides good test coverage, high-performance VPs also enable significant cost savings as they reduce the CI pipeline's runtime, saving compute resources and allowing developers to continue their work earlier by minimizing waiting time for CI results. This is depicted in Figure 1.

SIM-V addresses the need for high-performance RISC-V Loosely-Timed (LT) SystemC TLM-2.0 VPs. It is based on our custom Just-In-Time compilation (JIT) engine FTL, enabling high-simulation performance which we demonstrate in the case study of this work, outperforming QEMU by 2x. FTL is designed with SystemC TLM-2.0 in mind for easy integration into existing simulation environments. Furthermore, SIM-V is parallelizable allowing the SystemC TLM-2.0 standard compliant simulation of multiple RISC-V processors on multiple host processor cores. By integrating the FTL-based RISC-V processor model into our open-source (Apache 2.0) Virtual Components Modeling Library (VCML) [1], deep introspection and instrumentation in Python is enabled, as well as TLM transaction tracing, debugging with GDB, and VP control through a GUI.

II. Related Work

Using ISSs to test and verify target software before physical hardware is available is well established. Several ISSs for the RISC-V Instruction Set Architecture (ISA) exist. An overview that compares SIM-V to the existing solutions described in the following is provided in Table 1. The Spike simulator is a functional, interpretative ISS developed by the RISC-V community[2]. As it is an interpreter simulation performance is limited, but sufficient for verifying smaller, bare-metal target software. It is a standalone simulator and does not offer a SystemC TLM-2.0 integration. Therefore, integration into full system simulations is not easily possible.

Another prevalent simulator for the RISC-V ISA is Qemu[3]. It offers a very fast Dynamic Binary Translation (DBT) ISS framework: the Tiny Code Generator (TCG). This enables high simulation performance. Qemu is a monolithic simulator. The simulator target architecture is defined at compile time and cannot be changed. Therefore, it can only simulate one target architecture at runtime, so mixed architecture simulation, e.g. rv32 + rv64 or arm + rv64, is not possible. Furthermore, due to its monolithic architecture Qemu can not easily be integrated into an existing simulation. It does not offer a SystemC TLM-2.0 integration. Qemu allows to execute multi-core target systems in parallel on several host CPU cores through its MTTCG mode. It can also annotate time to target instructions in ICOUNT mode. However, both modes are mutually exclusive. Therefore, the user has to choose either higher performance through parallel execution or higher accuracy through instruction counting. To tackle some of the described issues, GreenSocs has released its QBox integration of Qemu[4]. By using QBox, Qemu can be integrated into a SystemC TLM-2.0 environment. However, due to the compile time defined target architecture, processor models of different architectures are executed in different host processes. This incurs an overhead. Also the limitations regarding ICOUNT and MTTCG mode still apply for each QBox instance.

	ISS Tech.	ISS C++ API	SystemC TLM-2.0 Compat.	Parallel Sim. with Timing Behavior	Combine Archs. (e.g. ARM + RH850)	Integrate into existing VP	Python CI Scripting	GUI
Spike	Interpret.	✅	❌	❌	❌	❌	❌	❌
QEMU	TCG	❌	❌	❌	❌	❌	❌	❌
QBox QEMU	TCG	❌	✅	✅	✅	✅	❌	❌
SIM-V	**Custom**	✅	✅	✅	✅	✅	✅	✅

Table 1. SIM-V feature matrix.

SystemC has become the de-facto standard simulation environment for VPs in large part due to its easy to use modeling primitives and its inheritance of the powerful C++ programming language. However, due to its single-threaded nature, SystemC has become the performance bottleneck for modern multi-processor designs. Various works exist that propose extending SystemC with capabilities to utilize multiple threads, processes or even host computers to accelerate simulation speed.

In the sc_during[5] approach, the authors propose the introduction of a new API call that allows execution of jobs on worker threads in an asynchronous manner while the main SystemC thread continues to operate. Such an asynchronous approach is also taken by SystemC Link [6], first dividing the simulation manually into segments and then spreading those over multiple processes or even hosts.

In contrast to these asynchronous approaches, Ventroux et al. [7] present a synchronous technique that enables parallelization only of activities that occur at the exact same point in time, yielding a more accurate timed simulator at the cost of reduced performance. Using extra tooling, simultaneous TLM quanta may be parallelized as well.

Finally, the authors of CoMix [8] present an approach that loosens the timing constraint of SystemC to achieve a better parallel utilization of the available host processors due to less synchronization overhead. Its lack of support for TLM DMI makes this technology a difficult choice for general purpose VPs, however.

III. SIM-V

A. Fast Translator Library (FTL)

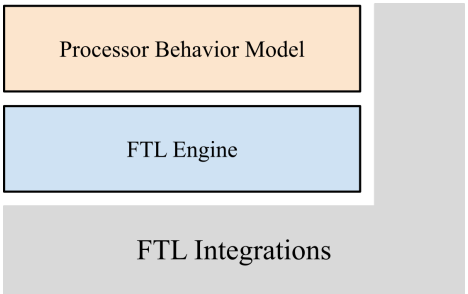

Figure 2. FTL architecture.

FTL is the origin of the high simulation performance of SIM-V. It enables rapid development of processor models, thanks to its intuitive API and powerful tooling. An overview of FTL's architecture is provided in Figure 2. Based on an instruction set description language, framework models can be generated that serve as a starting point for model development. Commonly required functionality, such as decoders, code generators, JIT caches is already present in the initial design, allowing engineers to immediately start working on the important part: the processor behavior model which describes the processor state and the behavior of its instructions. Instructions are described in an easy to use C++-based API, giving engineers maximum control and flexibility

when modeling their behavior. This process has been conducted for SIM-V, yielding one of the fastest processor models for RISC-V available today in record time.
The finished processor model offers a multitude of integration possibilities, for example as a standalone simulator for executing bare-metal programs, or as a SystemC module within a full TLM-2.0-based VP.

B. Integrating FTL Processor Models into Virtual Platforms

Even though a fast processor model is indispensable for high-performance VPs, it is only of limited use by itself. Therefore, it is important to provide free and open integration options to enable seamless integration of the processor model into an existing simulation environment. An overview of the available integrations for FTL-based processor models, such as the ones integrated in SIM-V, is provided in Figure 3.

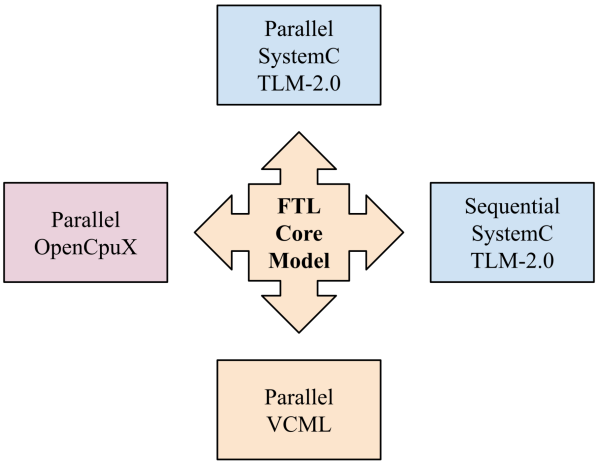

Figure 3. Open integration options for FTL-based processor models.

As FTL was designed with SystemC TLM-2.0 in mind, it is no surprise that it supports integration into standard Accellera SystemC TLM-2.0, which only enables sequential execution by default. Beyond that, a parallel integration with standard SystemC TLM-2.0 is provided, which in terms of technology is similar to the work presented in [5].

Besides the integration into standard SystemC TLM-2.0, an integration into the open-source OpenCpuX interface is provided, which enables the integration into non-SystemC as well as SystemC simulation environments[9]. This integration also allows for the parallelization of multiple target CPU cores on multiple host CPU cores.

Finally, an integration into our open-source SystemC TLM-2.0-based productivity library VCML is provided[1]. Besides the parallel execution of multiple FTL-based processors, it also offers other advantages. VCML comes with a GDB server for easy debugging of the resulting VP, which can also be controlled by a Python scripting interface. In addition, VCML provides deep instrumentation and analysis capabilities driven by our PyVP CI scripting framework. Through PyVP many points of VP execution can be instrumented such as reaching specific functions in target software execution, accessing specific registers and memory, or reaching pre-defined points in simulation time. From there the state of the VP can be manipulated, e.g. for fault injection testing. Furthermore, VCML supports tracing of all TLM transactions on the buses of the VP as well as attaching our open-source GUI ViPER to the VP for interactive use[10].

IV. Case Study

A. The SIM-V Virtual Platform

The SIM-V Virtual Platform embeds several FTL-based RISC-V processor models into multi-processor design with a rich peripheral ecosystem providing various memories, bridges, I/O controllers and coprocessors. It is intended to replicate the physical memory layout to QEMU's virt platform, allowing efficient reuse of existing, unmodified low-level software. The composition of SIM-V's major components is presented in Figure 4.

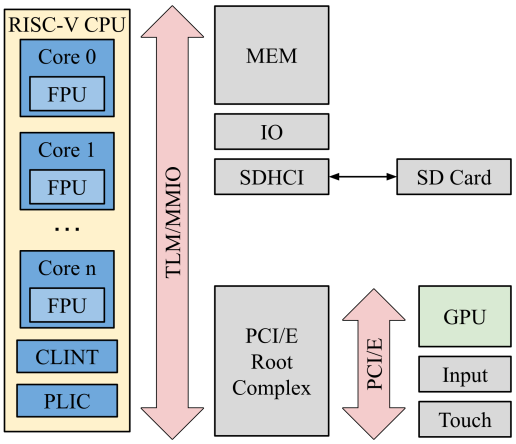

Figure 4. SIM-V VP Architecture.

While SIM-V can run any binary software compiled for the RISC-V architecture, it is most commonly used with a Linux kernel for rich operating systems and demanding 3d accelerated GPU graphic workloads. On the low end of the embedded spectrum, SIM-V targets the Zephyr real time operating system as well as plain bare-metal programs such as those that are commonly used with microcontrollers.
Interaction with the simulator is possible through a large variety of supported first and third-party tools, such as the ViPER GUI for inspecting the entire SystemC module hierarchy as shown in Figure 5. ViPER enables fine-grained control over the simulation progress and offers non-intrusive deep introspection into the details of each hardware model present in the design, such as MMIO registers, internal memories and configuration properties. While the simulation is running, ViPER offers convenient I/O terminals for all serial UARTs, as well as VNC-based screens for graphical login sessions.

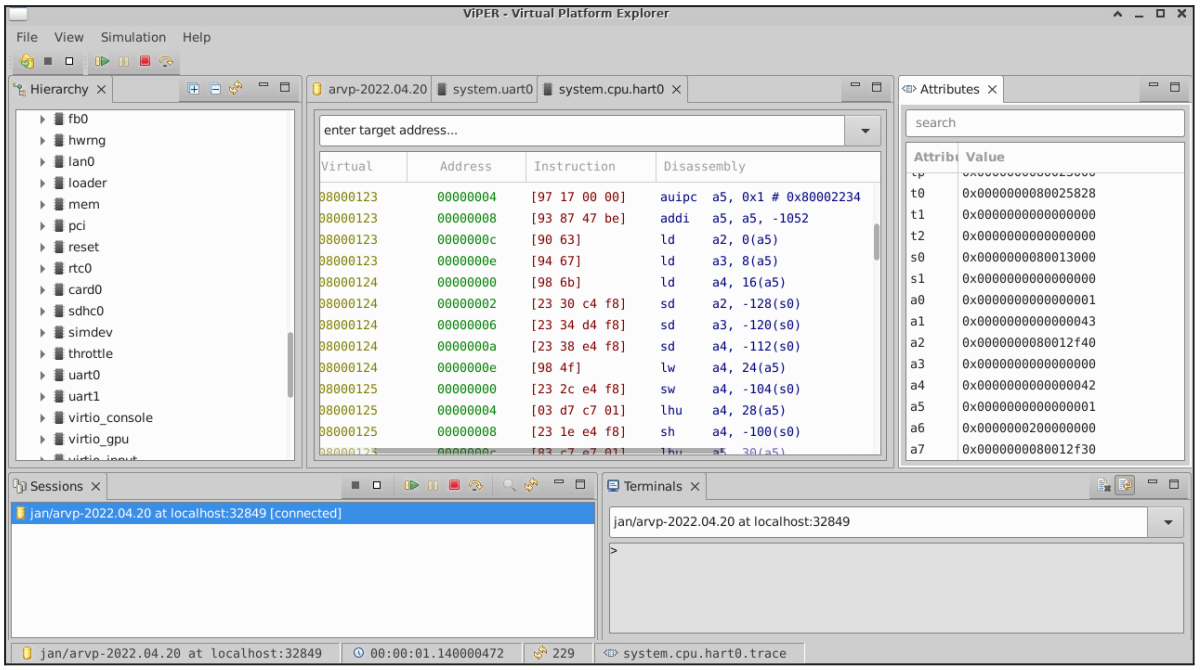

Figure 5. SIM-V ViPER GUI.

Furthermore, well known debugger integrations are supported, such as GDB. Given its widespread use in the industry, SIM-V's non-intrusive hardware-level debugger integration is expected to immediately feel familiar to most engineers, whether they are using GDB directly or via one of its many graphical frontends, such as Eclipse, for example.
Finally, SIM-V offers a rich scripting frontend for interfacing with the simulation from a Python environment. This is expected to be especially useful when operating SIM-V from a CI-environment, for example for testing

correct operation of a piece of software, e.g. a driver or a single interrupt routine. The Python script is capable of controlling simulation progress, investigating registers, setting breakpoints and investigating I/O from various peripherals, such as UARTs, Ethernet and CAN controllers. Finally, the scripting integration also enables fault-injection for improving test coverage of error handling code paths that are normally not executed.

B. Dhrystone Benchmark

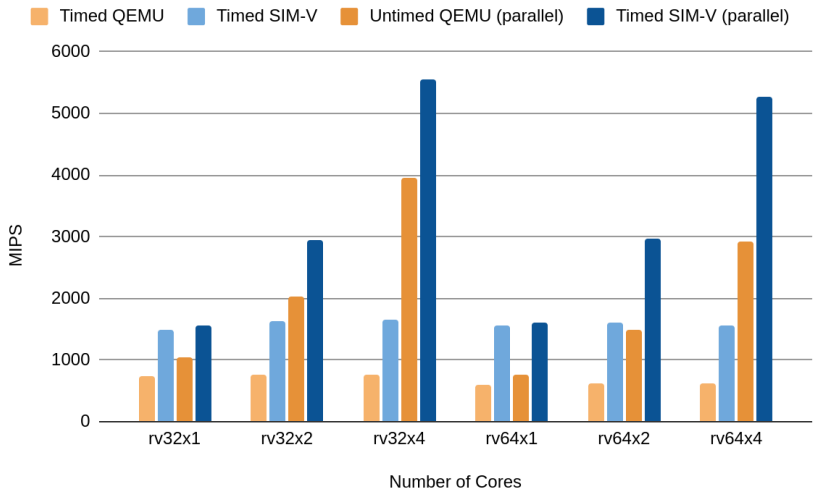

Figure 6. Dhrystone Benchmark SIM-V vs QEMU

To demonstrate the performance of our FTL-based SIM-V simulator, the dhrystone benchmark is executed for different RISC-V processor cluster configurations and simulation modes. For comparison, the byte identical binary is executed on the same configurations using Qemu. A Ryzen 5 3600 machine with 64GB of RAM running CentOS 7 was used as the simulation host. An overview of the MIPS performance results is provided in Figure 6. Here, the executed instructions on all processor cores of the simulation are divided by the required wall-clock time of the entire simulation. This yields the MIPS result for the entire simulation. The benchmarked RISC-V CPU cluster configurations are 32-bit RISC-V single-, dual- and quad-core, as well as 64-bit RISC-V single-, dual-, and quad-core. Two simulation modes are measured: sequential and parallel execution. It is important to note that Qemu is not capable of annotating time to instructions in parallel execution mode while SIM-V is. Hence, Figure 6 distinguishes between timed sequential execution and timed/untimed parallel execution. Furthermore, SIM-V carries a standard Accellera SystemC kernel, leading to a slowdown in comparison with Qemu.
Even though SIM-V is weighed down by the overhead of the SystemC simulation, it still outperforms Qemu by 1.97-2.64x in sequential execution. In parallel mode, SIM-V still outperforms Qemu by a factor of 1.41-2.13x, even though Qemu does not annotate timing while SIM-V does. Interestingly, Qemu performance deteriorates when simulating a 64-bit RISC-V system.

V. Conclusion

In this paper, we presented our RISC-V functional simulator SIM-V. SIM-V uses fast ISSs built using our custom JIT engine FTL. It was shown how FTL-based processor models may be integrated into existing simulation environments using a variety of free and open-source interfaces. By integrating with our open-source SystemC TLM-2.0 productivity library VCML, VPs with FTL-based processor models can be easily instrumented and analyzed. This is especially useful for CI, where these technologies enable significant cost savings by reducing CI pipeline runtime and thus minimizing developer idle times.
To substantiate this claim, we demonstrated the reachable performance of FTL-based processor models in a representative case study using our SIM-V simulator. For comparison, the widely adopted Qemu simulator was selected. Here, speedups of up between 1.41-2.64x were demonstrated.

References

[1] MachineWare GmbH, “Virtual Components Modeling Library (VCML)”, https://github.com/machineware-gmbh/vcml, accessed 01-05-2022

[2] RISC-V Foundation, “Spike RISC-V ISA Simulator”, https://github.com/riscv-software-src/riscv-isa-sim, accessed 01-05-2022

[3] Bellard, Fabrice. "QEMU, a fast and portable dynamic translator." USENIX annual technical conference, FREENIX Track. Vol. 41. No. 46. 2005.

[4] Delbergue, Guillaume, et al. "QBox: an industrial solution for virtual platform simulation using QEMU and SystemC TLM-2.0." 8th European Congress on Embedded Real Time Software and Systems (ERTS 2016). 2016.

[5] Matthieu Moy, “sc_during: Parallel Programming on Top of SystemC”, SystemC Evolution Fika, April 2022

[6] Weinstock, Jan Henrik, et al. "SystemC-link: Parallel SystemC simulation using time-decoupled segments." 2016 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 2016.

[7] Busnot, Gabriel, et al. "Standard-compliant parallel SystemC simulation of loosely-timed transaction level models: From baremetal to Linux-based applications support." Integration 79 (2021): 23-40.

[8] Sauer, Christian, Hans-Martin Bluethgen, and Hans-Peter Loeb. "Distributed, loosely-synchronized SystemC/TLM simulations of many-processor platforms." Proceedings of the 2014 Forum on Specification and Design Languages (FDL). Vol. 978. IEEE, 2014.

[9] Synopsys, “The OpenCpuX API”, https://github.com/snps-virtualprototyping/ocx, accessed 01-05-2022

[10] MachineWare GmbH, “Virtual Platform Explorer (ViPER)”, https://github.com/machineware-gmbh/viper, accessed 01-05-2022

[11] Weicker, Reinhold P. "Dhrystone: a synthetic systems programming benchmark." Communications of the ACM 27.10 (1984): 1013-1030.