Open source verification environment for RISC-V

Josep Sans[1], Alberto Moreno[1], Àlex Torregrosa[1] and Roger Espasa[1]

[1]Semidynamics

Abstract

The paper presents a fully open source verification environment to test and debug RISC-V cores and the IPs that compose them. The verification environment is built around Verilator, and extended with different open source libraries to complete a fully UVM-compliant verification environment, which can be run in any platform, without requiring any license. This allows high parallelism, which (1) speeds up the execution of the regression set, and (2) enables usage of a resource-hungry coverage driven verification methodology. Furthermore, Verilator's code translation style from RTL to C++ also enables the use of fuzzing techniques for the DUT, which further helps increase overall coverage.

Introduction

In contrast to the software world, where most development tools are open-source and freely available, in RTL projects all the necessary tools that allow to simulate and functionally verify the RTL code require expensive commercial licenses. The lack of open-source tools in this field makes it very expensive to acquire a sufficient number of licenses to parallelize the amounts of simulation hours to properly test and verify large RTL designs.

Even though there are license free simulators like ModelSim[1], or open-source simulators like Icarus Verilog[2], they lack the sufficient language features to build complex test benches to fully verify non-trivial RTL projects like a floating point unit, a memory pipeline, or a RISC-V core.

Thankfully there is another tool that takes a different approach to how to simulate an RTL design: Verilator[3]. Verilator transpiles the SystemVerilog RTL code into C++, generating a functional equivalence model of the RTL design. Later on, this new model can be extended with code that composes the testbench, resulting in a complete verification environment that does not require any license to run. The only limitation is the number of CPU cores available in order to run simulations.

This paper explores the viability of Verilator and different open-source libraries in order to build an open-source environment to verify a RISC-V core. It will also talk about the different verification approaches we have used for taking advantage of the verification environment described in the paper, in order to verify our RISC-V cores and their internal modules.

The software stack

In order to build up the open source verification environment we will use the following tools: Verilator, UVM-SystemC library, FC4SC[1] library[4], Spike[5] and all the existent C++ tools that are generally available (compilers, debuggers, performance analysis, ...). Figure 1 shows the integration of the different tools and libraries.

Verilator

Verilator is an open-source tool that is able to transpile RTL code written in Verilog or SystemVerilog into a C++ model that can be later compiled and executed. Since Verilator is more a compiler than a simulator, the generated model can be instantiated in a wrapper along with the test bench code having as a result an executable containing the RTL functional model and its test bench. It is also a good alternative to commercial simulators as the code generated it usually runs faster.

UVM-SystemC

UVM is one of the more widely adopted verification libraries in the industry. The port of the UVM library into the System-C runtime enables the use of the same verification concepts already established and to reuse all the existing knowledge in order to build the test benches.

FC4SC

One of the big handicaps Verilator has is the lack of integrated coverage analysis tools. Currently, Verilator only provides toggle coverage information. To provide support for functional coverage in the testbench the FC4SC library can be used in order to be able to

[1] Functional Coverage for System-C

replicate the coverage directives the SystemVerilog language has.

Spike

Spike is one of the most up to date RISC-V open-source simulators with the latest ratified RISC-V extensions. Although it does not provide a public API, the simulator code can be easily modified to create a set of endpoints to interact with Spike from the test bench. This allows creating a co-simulation environment with Spike and the RTL design, checking at each clock that the design under test (i. e. the RISC-V core) internal state is correct.

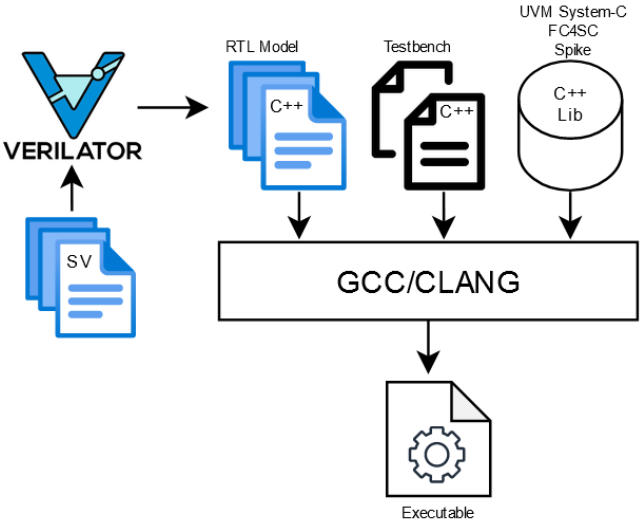

Figure 1: *Verilator software stack*

RISC-V core use cases

For our RISC-V cores verification, we use a coverage-driven random test generation methodology [6]. While this methodology is great to explore corner cases of the design, it is resource hungry. Our methodology uses a genetic algorithm that creates a pool of tests (i. e. the population), runs them on the simulator, obtains coverage metrics, and uses this metrics to create a new pool of tests based on crossover of the previous generation tests. Best results are obtained when the population is large; however a large population requires running a large number of test cases, which is either expensive or slow in commercial simulators. This scenario is ideal for the open-source Verilator based verification environment, since allows us to greatly parallelize the tests runs, thanks to the fact that we are not limited by the number of licenses in order to run the simulations.

Another benefit we obtain from using Verilator is that it allows using a fuzzing technique (a common technique in the software world) to generate tests that increase coverage[7]. Since the generated C++ is a direct translation of the SystemVerilog code, its coverage is highly correlated to the RTL code coverage. This has allowed us to use state-of-the-art fuzzing tools, like AFL++[8] to drive the constrained test generation for some internal modules of our RISC-V cores. These tools tend to produce minimum-size tests, which are easier to debug, and, in the case of AFL++, includes scripts to triage and minimize tests.

Conclusions

In this paper, we propose an open-source verification environment for RISC-V cores and RTL IPs that yields the best simulation cycle per cost ratio. All the mentioned strategies have been used by us in order to verify industry level RISC-V cores, both at top and unit level.

As the open-source tools presented in this paper gain momentum, the more contributors will appear, and the more completed and up to spec the tools will be. As an example, Verilator, already has support for event driven simulations, and in the near future, there will be support for native SystemVerilog UVM, making Verilator an excellent tool to run large numbers of regressions without needing to adapt the test bench using the UVM-SystemC library.

References

[1] Siemens. *RTL and gate level behavioral simulator*. URL: `https://eda.sw.siemens.com/en-US/ic/modelsim/`.

[2] Stephen Williams. *ICARUS Verilog Compilation System*. URL: `https://github.com/riscv-software-src/riscv-isa-sim`.

[3] Wilson Snyder. *Open source SystemVerilog simulator and lint system*. URL: `https://www.veripool.org/verilator/`.

[4] *Functional coverage for System-C*. URL: `https://github.com/amiq-consulting/fc4sc`.

[5] Riscv-Software-Src. *RISCV-software-src/RISCV-isa-SIM: Spike, a RISC-v isa simulator*. URL: `https://github.com/riscv-software-src/riscv-isa-sim`.

[6] F. Corno, F. Cumani, and G. Squillero. "Exploiting Auto-adaptive μGP for Highly Effective Test Programs Generation". In: *Evolvable Systems: From Biology to Hardware*. Ed. by AAndy M. Tyrrell, Pauline C. Haddow, and Jim Torresen. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 262–273. ISBN: 978-3-540-36553-2.

[7] Timothy Trippel et al. "Fuzzing Hardware Like Software". In: *USENIX Security Symposium*. 2021.

[8] Andrea Fioraldi et al. "AFL++: Combining Incremental Steps of Fuzzing Research". In: *14th USENIX Workshop on Offensive Technologies (WOOT 20)*. USENIX Association, Aug. 2020.