2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE 10 YEAR ANNIVERSARY

Scalable agile processor verification using SystemC UVM and friends

Eyck Jentzsch, MINRES Technologies GmbH, Munich, Germany (*eyck@minres.com*)

Stanislaw Kaushanski, MINRES Technologies GmbH, Duisburg, Germany (*stas@minres.com*)

This tutorial focuses on using SystemC and the UVM methodology to verify a processor core. UVM is a powerful verification framework that allows for the creation of testbenches that generate constrained random stimuli, track functional coverage and verify designs. SystemC UVM by its nature allows to easily integrate C/C++ reference models which makes it an obvious choice for the task of verifying processor cores.

RISC-V is an open-source instruction set architecture designed to be simple, extensible and easy to implement. As such, it is becoming increasingly popular in the industry and academia. It has gained widespread adoption due to its open-source nature, configurability, and scalability. There is a growing need for highly configurable RISC-V cores that can be customized for specific use cases and applications in the IoT and Edge domain. However, the complexity and variability of these cores make verification a significant challenge, especially when it comes to verifying custom instructions. To ensure these cores is correctly implemented and functioning, it is essential to verify it using various stimuli and functional coverage metrics.

In this tutorial, we will provide an overview of the SystemC UVM methodology and demonstrate how to use it to verify RISC-V cores. We will focus on the following topics:

1. Introduction to SystemC UVM and its benefits for verification
2. Overview of the TGC RISC-V core family and CoreDSL
3. Creating a SystemC UVM testbench for a RISC-V core
4. Generating constrained random stimuli to exercise the core
5. Defining functional coverage metrics to measure the quality of the verification
6. Using a C++ reference model of the RISC-V core to compare against the DUT
7. Running the verification using different open source and commercial simulation tools and analyzing the results

By the end of the tutorial, participants will have a comprehensive understanding of how to apply SystemC UVM to verify a processor core using constrained random stimuli, functional coverage and C++ reference models. They will also gain an appreciation for the benefits of using SystemC UVM for such kind of verification tasks, enhance their knowledge base in verification methodologies and strengthen their skills for future verification projects.

Throughout the tutorial the MINRES family of RISC-V cores named TGC will be used as devices under verification. The cores are defined using CoreDSL, a domain specific language to describe instruction sets in a simple and comprehensible way. This allows also to easily extend the cores with custom instructions. The TGC family itself consists of a set of basic configurations with varying numbers of pipeline stages, registers, instruction sets and custom extensions. Moreover, each basic configuration can be customized and extended with additional features such as different bus interfaces, interrupt controllers, memory protection, caches etc.

This high level of configurability in conjunction with functional safety requirements implies a highly automated, modular, and reusable verification environment using the CoreDSL specification as single source of truth to defining random stimuli generation, coverage collection, and coverage goals.

We will show how SystemC UVM can be used to implement this modular verification environment integrating an instruction set simulator as reference for the behavior of the cores under verification. The environment leverages FC4SC to access functional coverage and guide the constrained randomization for which CRAVE and other open-source libraries are being used. The tight integration in the UCM testbench allows for generation of endless instruction sequences without a forward progress guarantee. This allows for verification of more corner cases which is usually a limitation of other well-known approaches.

The tutorial will also focus on how to hook-up the verification environment with the simulator running the RTL. The modular nature of the UVM environment allows to use Verilator which allows for fast and (virtually) unlimited parallelization of the simulation runs especially in a continuous integration setup. Without modifications of the UVM tests, it is also possible to connect to commercial Verilog Simulators using UVM Connect or even to a hybrid FPGA simulation solutions like the MINRES' Raven product.

Another important aspect covered in the tutorial is the use of open-source components like SystemC UVM, FC4SC, PyUCIS, Crave, SystemC components library and many others to ease verification of RISC-V cores in an agile and test-driven way. By reusing these components, the effort to create and maintain a verification environment is significantly lowered. Additionally, it becomes easier to use the UVM environment in a continuous integration environment, e.g. using Jenkins, as the dependencies to proprietary tools shrinks.