Unlocking open source RISC-V SoC verification

RISC-V Week, Paris, 2022-05-05
Michael Gielda, mgielda@antmicro.com
ANTMICRO: COMMERCIAL OPEN SOURCE SUPPORT & DEVELOPMENT SERVICES

• Much of Antmicro’s work revolves around open source FPGA and ASIC tooling development where we work with partners such as Google, Western Digital, Microchip and SiFive

• We develop tools for synthesis, place and route, linting, formatting, simulation & co-simulation, hardware-software co-design, design verification & automation

• These tools allow us to effectively build future proof systems, CI-driven flows and enable parallelization of development for our customers
ANTMICRO & CHIPS ALLIANCE

- Like Antmicro, CHIPS Alliance believes that on top of RISC-V, open source silicon requires un-core IP and a vibrant ecosystem of open source tools.
- CHIPS coordinates with RISC-V closely since RISC-V is our de-facto primary ecosystem.
- CHIPS believes open tools are possible and necessary for unconstrained collaboration, unlocking software-driven workflows & AI-assisted ASIC design.
- We’re hosting OpenROAD/OpenLANE/OpenFASoC.
- Home to the open source F4PGA toolchain, the FPGA Interchange Format.
- Home to Chisel and many other HDL-related tools.
- Home to our open source verification efforts!
WHY DO WE NEED OPEN SOURCE UVM / SYSTEMVERILOG SUPPORT?
While RISC-V is spawning new approaches to HW design which offer tangible improvements, there is an immense body of pre-existing work. Sustainable results require an inclusive approach towards existing pool of designs and designers. We need to make it easy to combine and remix methodologies.

Important progress has been made in verification space - and we’re now analyzing how to deploy this to production for some limited use cases! (this still means a lot of work ahead of us)
WHAT WOULD OPEN SOURCE UVM/SYSTEMVERILOG SUPPORT ENABLE?

- Combining commercial ecosystem with open source tools and methodologies
- Chip-making companies can benefit from open source while keeping their existing UVM codebase
- Lack of licensing limitations would allow scalable, reproducible CIs
- Number of open source cores and lots of pre-existing IP implemented in SystemVerilog, e.g.
  - SweRV
  - Ibex
  - BlackParrot
  - Core-V
- Building a collaborative ecosystem around ASICs
- Proving open source tools are really possible
HOW TO GET THERE?

- Identify missing functionalities and features
- Reuse existing solutions
  - There are many existing projects which can be improved
- Create well documented and transparent flows
  - Include automated tests and status reporting in projects
- Cooperate with others
  - Gather information on what is needed
- Provide incremental value
WHAT IS MISSING?

Unlocking open source RISC-V SoC verification
We created a test suite to determine the SystemVerilog support level in various open source tools.

- Aims to pinpoint all the supported and missing SystemVerilog features in various tools.
- Generates report from last passing master build at [github.com/chipsalliance/sv-tests](https://github.com/chipsalliance/sv-tests).
- Introduces three types of tests:
  - Testing individual features as per the SystemVerilog standard.
  - Existing third party test suites.
  - Selected open source IP cores, such as SweRV, Ibex and others.

**CHECK THE SPEC: SV-TESTS**
GET AN EVENT DRIVEN SIMULATOR

• One of the biggest missing pieces in open source open source UVM is a fast, time aware and event driver simulator
• Verilator was the obvious choice: it’s the fastest simulator available on the market
  ▫ no time awareness and events support though!
• To address this we decided to reimplement the way Verilator handles simulation scheduling
STRATIFIED SCHEDULER

• SystemVerilog simulation time is divided into time slots which are further divided into regions
• Different types of statements fall into different regions
  ▫ Normal assignments are in Active,
  ▫ Non-blocking assignments are in NBA,
  ▫ Concurrent assertions are in Observed, etc.
• These regions are not simple divisions of time; the simulation can go back to an earlier region within a time slot, e.g. if a non-blocking assignment triggers an active block
• Some events can get delayed to a following time slot
So far, Verilator only partially implemented a stratified scheduler, but only by statically ordering the generated code. That is not sufficient in cases where we cannot predict (during compilation) what events get scheduled or delayed and when it happens (such as delays or forks in a deep call stack, or in a virtual function). UVM requires a more dynamic approach that allows scheduling events at runtime.

DYNAMIC SCHEDULING
OUR APPROACH: COROUTINES

- Complete rewrite of the way Verilator schedules events (in several iterations)
- We use C++ coroutines (from C++20!) to handle the scheduling and synchronization of asynchronous events in the simulation
  - Coroutines are a form of cooperative multitasking – and that’s fine for this case!
  - With coroutines, we don’t have to worry about multithreading (as much)
  - The overhead for coroutines is significantly smaller than threads
- Read more about the technical details in our blog note
WHAT IS ALREADY POSSIBLE

- Delays
- Event variables
- Fork / join
  - join (wait for all children to finish)
  - join_any (wait for any one child)
  - join_none (do not wait, continue the main process immediately)
- Wait statement
- Constrained randomization
  - Not related to scheduling, but needed for UVM
  - Only basic constraints are supported so far; the next step is to add support for a real solver
DYNAMIC SCHEDULER EXAMPLES

- We’ve created a public GitHub repository showing example usage of the new features we added to Verilator: [github.com/antmicro/verilator-dynamic-scheduler-examples](https://github.com/antmicro/verilator-dynamic-scheduler-examples)
- The CI there runs simulations of all the examples in the repository

Verilator with a dynamic scheduler

Copyright (c) 2021-2022 Antmicro

This repository contains a number of examples that showcase our attempt at implementing a dynamic scheduler for Verilator, which can be found here, as well as limited support for [randomize] constraints (available here).

This version of Verilator requires GCC 10 or newer, or Clang (tested with Clang 13).

After cloning, please run:

```
git submodule update --init
```

You can run these examples using `make`:

```
nake EXAMPLE
```

where `EXAMPLE` is the name of one of the directories in `examples` (listed below).

Available examples

- `uart` – the biggest and most interesting one, it’s a testbench for a UART transmitter and receiver,
- `clock` – generation of a clock in an `initial` block using delays,
- `events` – event triggers, event controls, events in sensitivity lists,
- `fork` – a showcase of the `fork` functionality, along with all possible `join` types,
- `pong` – two `initial` blocks sending events to each other over multiple timeslots,
- `randomize` – demonstrates support for the [randomize] class function with constraints,
- `wait` – shows the `wait` statement in action.

New functionality
UHDM INTEGRATION

- **Surelog** is an open source SystemVerilog 2017 Pre-processor, Parser, Elaborator and UHDM Compiler
- Universal Hardware Data Model (**UHDM**) is used to exchange the information about elaborated SV design between the parser and other tools
- The Surelog->UHDM flow allows fully open source synthesis of SystemVerilog code thanks to the Yosys **SystemVerilog plugin** and fully open source SystemVerilog verification thanks to a custom **Verilator** frontend
- Already in late 2020 we could e.g. parse, synthesize and simulate OpenTitan’s Ibex core directly from the SystemVerilog source
OPEN SOURCE IBEX SYNTHESIS AND SIMULATION IN VERILATOR/YOSYS VIA UHDM/SURELOG

SystemVerilog design → Surelog → Verible? → Another parser? → Design expressed in UHDM → libuhdm

Another tool? → libuhdm

Yosys

Verilator
VERILATOR 5

- By introducing the concept of delayed execution to Verilator we have enabled:
  - Event driven simulations
  - Full Verilog/SystemVerilog testbenches (instead of C++ testbenches)
- Stratified scheduling now being integrated into mainline Verilator
  - Coming soon as Verilator 5!
  - Will be released together with major changes regarding the static scheduler (assigning statements to actual stratified scheduler regions and executing those regions in an order specified by the Language Reference Manual)
WHAT ELSE NEEDS TO BE DONE?
NEXT STEPS

• Extending Verilator with support for more SystemVerilog features used in UVM such as:
  ▫ Clocking blocks
  ▫ Assertions
  ▫ Class parameters
  ▫ Cyclic randomization
  ▫ Unpacked structs
  ▫ Functional coverage

• Runtime optimization

• Integration with UHDM work

• Identifying issues preventing integration with verification test suites used by different projects like e.g. riscv-dv and addressing them
FUTURE GOALS

• Long-term goal is to enable full UVM support in Verilator
  ▫ We are now looking at real life production grade designs to gauge what more is required to run their complete validation
  ▫ If you want to get a study of coverage for your own use case, let us know
  ▫ One other obvious focus are open source designs like SweRV or OpenTitan
• An open source flow for verification should lead to more open source verification testbenches and IP being shared!
VHDL SUPPORT

• Evaluating complexity of effort to extend the flow with full VHDL support
  ▫ The plan is to reuse existing UHDM frontends in the tools
  ▫ We would need to extend existing VHDL parser/elaborator with UHDM generation feature
• Considering a few parser/elaborator candidates for first implementation
• Would allow for interesting use cases for event-driven Verilator
• Looking for interested parties!
SYSTEMC SUPPORT

- Intel, already a CHIPS member, reached out to contribute their SystemC compiler
- The tool can generate SystemVerilog, meaning that interoperability is possible without significant effort
- We could also go directly to UHDM, which might yield better processing speed/results, but the “easy” approach looks feasible
- This would provide us with pretty broad coverage of “traditional” methods of describing hardware
- Let us know if you are interested in this effort
THANK YOU FOR YOUR ATTENTION!