RISC-V Core FPGA tracing for verification

Alberto Moreno[1], Jordi Cortina[1], Àlex Torregrosa[1] and Roger Espasa[1]

[1]Semidynamics

Abstract

This paper introduces an FPGA verification, debug and performance platform for RISC-V cores. The system utilizes the open source RISC-V simulator Spike as a golden model that can be co-emulated with the core in real time. In our experience, this enables verification at up to 50 MHz. This allows for rapid iteration of new features by testing them in a realistic environment without a previous need for unit-level verification. The platform may also be configured to act unobtrusively in order to generate real time accurate performance metrics.

Introduction

One of the most challenging aspects when designing a RISC-V core is guaranteeing its functional correctness. The industry has long settled into using unit-level verification of all the components present in the core, combined with the use of an FPGA-based emulation platform such as Veloce[1] or Zebu[2].

Unit-level verification using simulation tools ensures high visibility into the DUT, as well as allowing fine-grain control of its randomization. Typically, several unit-level testbenches are combined into "sections" of the core, which are exercised intensively with random testing, sometimes called super-unit level testing.

Unit and Super-unit level testing have one major drawback: slow speed. Since they are based on simulation tools, the verification team is limited by a combination of the number of CPUs available to the verification team, the number of simulation licenses available, and the speed of the CPUs. Hence, for high volume verification, most teams turn to the use of FPGA-based emulation systems.

When using FPGA emulation, one needs to ensure that the RTL running on the FPGA performs correctly. In commercial emulation systems, the UVM testbench can be run in parallel with the RTL, checking correctness as if one was using simulation. The major downside of this approach is that, despite using an FPGA, speed may become limited by the testbench.

In this paper, we introduce an alternative method for verification using FPGAs that allows high speed (up to 50Mhz in our experience) while still checking the correctness of the RTL by executing in parallel, or co-emulating, a Simulator along the RTL.

FPGA co-emulation overview

The co-emulation platform is composed of an FPGA, a RISC-V simulator and the tracing platform.

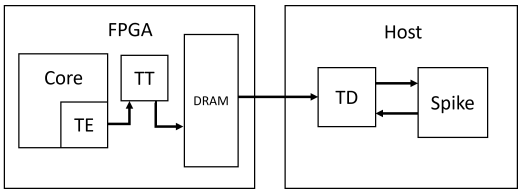

Figure 1: *Co-emualtion platform overview*

The FPGA is the physical platform in which the RISC-V core will be synthesized. It must be large enough to fit the core, but in recent years, FPGAs with capacity to emulate large amounts of cells have been developed and released by the two major vendors Xilinx and Altera. This significantly raises the bar for which cores can be tackled by this approach. It is important that the selected FPGA has access to a large block of DRAM in order to store the debug information that will be generated by the tracing platform.

The simulator is the tool that does the heavy lifting for the verification effort, acting as a golden model for the RISC-V spec. In our implementation we use a lightly modified version of the open source Simulator Spike[3]. These modifications are performed in order to accommodate the particularities of the design core, but the simulator remains otherwise unchanged.

The tracing system is endeavored in capturing information from the core within the FPGA, pacing the simulation and verifying that core and simulator match. It is composed of three subsystems: the Trace Encoder (TE), the Trace Transmitter (TT) and the Trace Decoder (TD). Figure 1 shows an overview of the platform, with all its different parts.

Trace Encoder: The Trace Encoder is a piece of RTL that is synthesised within the core itself. It is connected to the relevant signals for the core, which are encoded into packages and sent to the TT. The interface between the TE and the TT must be wide enough to accommodate for the signals being traced.

Trace Transmitter: The Trace Transmitter is a module that is synthesized outside of the core, but still

in the same FPGA. It is connected to the TE within the core and to the TD in the host machine. The TT has access to a dedicated address space in a DRAM in order to store and transmit traces towards the TD. Another important role of the TT is flow control: there must be a mechanism to stall the core whenever the allocated DRAM runs out of space to store traces. Because of this, the relationship between the size of the traces and the bandwidth of the connection between FPGA and host will impose an upper limit on the performance of the co-emulation.

Trace Decoder: The Trace Decoder is a piece of software that runs in the host machine. It must have outside access to the DRAM address space where the TT stores traces, which the TD then fetches and interprets. It must then interface with the Simulator to execute the same instructions and verify the results.

As an example, the TE may connect to the retirement signal for the core, indicating the Program Counter (PC) of every instruction that is being retired. It may also include the result of executed instructions, such as an addition or a load. This information is sent to the TT, which will store it in its dedicated address space. The TD will then fetch this information and use it to infer that an instruction has been retired in the core. This is used to indicate the simulator that a new instruction must be executed. The PC and the result of this instruction can then be compared against the one reported by the core. If a mismatch is detected, the TD can inform of the PC and the differences in the result.

Even if the information transmitted contains only the PC and the result, the simulator can extrapolate much more information. In the previous example, it is possible to infer the data contained in all the integer registers in the core by just looking at the data contained in the simulator's registers. It's also possible to generate a trace of all the executed instructions and make deductions about the internal state of the core.

Advantages and challenges

We successfully tested this strategy in the design of a commercial core. The design can be quickly tested in a real environment, running binaries in a modern OS while still obtaining feedback about the correctness of the execution. This enables a rapid iteration cycle in which new features can be tested in a system-wide platform at up to 50 MHz. Errors are observed much faster, without the need for producing extra verification IP tailored to every new module. Furthermore, this platform may be included into a Continuous Integration strategy where every significant code update can be automatically verified in an FPGA.

An additional advantage of our method is that it

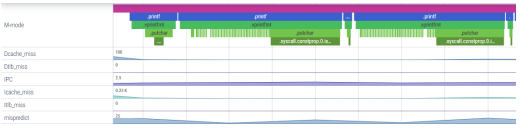

Figure 2: *Performance trace*

allows cycle-accurate performance analysis. If the information transmitted is kept below the connection bandwidth between core and host machine, this tracing system can run without any stalls. This enables real-time monitoring of a large number of performance metrics as well as off-line execution traces. It can show, for example, number of misses and hits for a given executed function. Figure 2 depicts an example of one such trace, showing how different metrics evolve in time, including the call stack and the name of the different functions being executed.

But this approach does not come without challenges. The bandwidth between the FPGA and the host machine is often a bottleneck, particularly when too much information is being tracked. This forces a trade off between the amount of information that can be verified and the speed at which tests can be run.

Ultimately, the major challenge for this approach is reproducibility. Using an FPGA with real DRAM enables testing in an environment that is very similar to post-silicon. This may include large variability in between runs. When an issue is detected in co-emulation, it is sometimes complex to reproduce the conditions that caused it. Thus, several re-runs of the same tests are often required in order to reproduce it.

Conclusions

In this paper we propose a verification strategy for the design of RISC-V cores that has been tested in a commercial design. This strategy includes the use of FPGAs to co-emulate cores with an open source RISC-V simulator. The platform implemented for this use has been successful at accelerating the iteration speed between implementation and testing of features in a realistic environment. The main challenge for this approach lays in reproducing issues found while in co-emulation. But this is attenuated by the speed at which tests can be re-executed.

References

[1] Siemens. *Veloce Prototyping - FPGA*. URL: `eda.sw.siemens.com/en-US/ic/veloce/fpga-prototyping`.

[2] Synopsys. *Zebu Server Emulation System*. URL: `www.synopsys.com/verification/emulation/zebu-server.html`.

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