embeddedworld2020
Exhibition&Conference
...it's a smarter world

Impact of RISC-V Adaptability on SoC Verification Methods

Simon Davidmann, Lee Moore and Larry Lapides
Imperas Software Ltd.
Oxford, United Kingdom
larryl@imperas.com

***Abstract*— As the RISC-V Instruction Set Architecture (ISA) matures, and SoCs are developed using RISC-V, there is a need to address the new verification challenges of RISC-V based SoCs. For SoCs built using traditional processor cores, the verification tasks are well known, as the starting point is based on the assumption of "known good IP." The new verification challenges include verification of the RISC-V processor IP; verification of the processing element (PE) containing the RISC-V core(s) (especially relevant in SoCs with a fabric designed for AI processing); connection of the processor itself or the PE to the network on chip (NoC) and multiple PEs communicating through the NoC to each other. In this paper, the verification challenges for RISC-V SoCs are presented. Specific verification flows including new test and instruction stream generators, reference models and metrics are presented in detail including the results of using these flows on real processor IP and SoCs.**

Keywords—RISC-V, Verification, Processor, SoC,

I. INTRODUCTION

As the RISC-V Instruction Set Architecture (ISA) matures, and SoCs start to be developed using RISC-V, the development community needs to start addressing the challenges of RISC-V based SoCs. Chief among these challenges are those related to verification.

For SoCs built using traditional processor cores from vendors such as Arm and MIPS, the verification tasks are well known. The starting point for verification is based on the assumption of processor IP that has been exhaustively tested, and is assumed to require no additional verification when received from the processor IP vendor. There is still a lot of work to be done to verify a SoC to the point of acceptability for tape out, and the scalability problems for dealing with the size and complexity of the full SoC are still there, but these are known issues.

With RISC-V there are possibly four new verification challenges to address for SoC projects: 1) verification of the RISC-V processor IP; 2) verification of the Processing Element (PE) containing the RISC-V core(s) (especially relevant in SoCs with a fabric designed for AI processing); 3) connection of the processor itself or the PE to the network on chip (NoC); and 4) multiple PEs communicating through the NoC to each other.

For the processor core verification, three different scenarios exist: the processor IP (RTL) can be purchased from one of the many processor IP vendors in the RISC-V community, the processor IP can be downloaded from one of the open source repositories or the SoC developer can build the processor RTL from scratch. In the first two situations, there will have been a significant amount of verification performed on the processor IP; however, it is almost a certainty that the processor IP will have less verification and less maturity than a core from a traditional processor IP vendor. Also, if custom instructions are added to the core, no matter the source of the original RTL the core needs to be thoroughly verified for both the new features and to confirm the original base core quality has not been compromised.

In many AI (Artificial Intelligence) architectures, the design is structured in a hierarchy with a processing element consisting of one or more CPUs, plus AI accelerators or co-processor(s), plus some additional logic to connect to the SoC AI fabric. This is a critical feature to support the desired applications,

and offers convenient abstraction levels to align the verification methods.

Next, while the processor IP and PE have been verified, and the NoC has been verified (assuming that an existing NoC IP is used), the interaction of the RISC-V processor, PE and the NoC is unique to the design and requires verification.

Last, while the verification of a single PE is needed, verifying multiple PEs working with each other through the NoC is also needed. This is especially true in the case of designs based on RISC-V, since the Open ISA flexibility allows for optimization of each of the cores, so all the various combinations of PEs will need verification as well.

These verification challenges can be addressed within the general framework of UVM verification methodology and tools, however, some innovation is needed, along with collaboration between processor IP vendors, EDA vendors, other tool developers and the RISC-V SoC developers. For example, several test generation and instruction stream generation tools have been developed to address RISC-V specific requirements, and new directed test suites have been developed for specific RISC-V extensions such as the vector instructions. New reference models are needed for the RISC-V processors and PEs. New metrics are needed, especially for the processor and PE verification areas, perhaps such as instruction coverage. Flows with these tools need to be robust to handle the variety of processor IP scenarios elaborated above. Plus, a robust flow is needed to ensure that a solitary "bad actor" does not insert a backdoor into the processor or SoC.

Other areas that are being looked at to address RISC-V SoC verification include using hybrid emulation-virtual platform systems for hardware-software co-verification, using Portable Stimulus (PSS) for multiple PE and full chip verification, and using the nature of the AI algorithms to constrain the SoC state space.

In this paper, the verification challenges for RISC-V SoCs are discussed and an overview given of potential solutions. Specific verification flows including new test and instruction stream generators, and reference models and metrics, are presented in detail including the results of using these flows on real processor IP and SoC designs.

II. Compliance is not Verification

With RISC-V, as an open ISA specification [1], any implementation will need to be tested against the latest RISC-V compliance suite.

The objective of the compliance process is to ensure that implementations are correctly following the specifications, with the expectation that compliant devices will exhibit sufficient compatibility to leverage the emerging ecosystem for tools and software. Put more simply, compliance is confirming that the designers have understood the specifications. Since the ISA specification does not include details of microarchitecture, differences in device performance and application focus are expected and of course permitted.

Since the compliance tests use expected functionality as the basis of the test suite, this incurs an overlap with some aspects of Design Verification (DV). However, the compliance suite is not exhaustive for all functionality and is focused purely with the structural specification aspects of the ISA, i.e. compliance is a subset of DV.

The RISC-V Compliance Suite is developed within the RISC-V Foundation Working Group on Compliance ("Compliance WG"), and the latest test suites are available from the RISC-V compliance GitHub repository [2].

III. Custom Instructions and Reference Models

A reference model is a key to processor-related DV tasks. This is usually an instruction accurate (IA) model of the processor, often called an Instruction Set Simulator (ISS). The Compliance WG GitHub repository includes the riscvOVPsim ISS. The riscvOVPsim simulator implements the full and complete functionality of the RISC-V Foundation's public Unprivileged (formally known as User) and Privilege ratified specifications. The simulator is command line configurable to enable/disable all current optional and processor specific options in the RISC-V specification. The simulator is developed, licensed and maintained by Imperas Software Ltd., and is fully compliant to the Open Virtual Platforms (OVP) [3] open standard APIs. Most recently, support for the vector and bit manipulation instructions were added to the OVP RISC-V processor models.

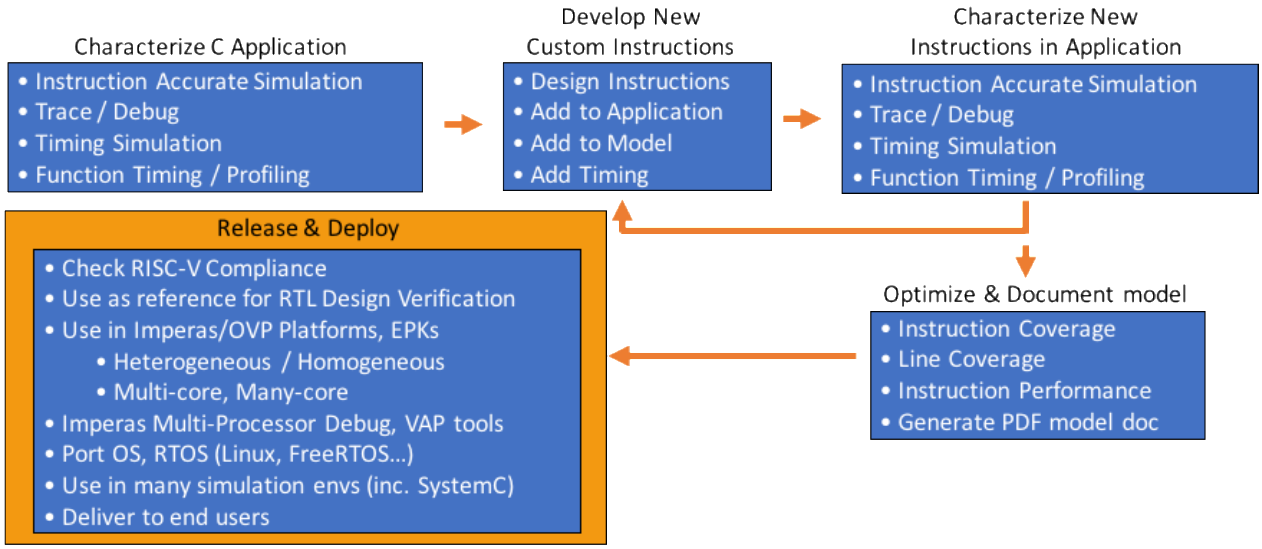

Fig. 1. Flow for adding custom instructions to the RISC-V processor model.

When custom instructions are added to the RISC-V processor, those instructions need to be added to the reference model. Imperas has previously developed a methodology for adding and optimizing custom instructions [4]. This flow is shown in Fig. 1. The resulting model, which includes both the standard RISC-V instructions and the custom instructions, can then be used as a reference model for DV of the processor RTL.

IV. Processor IP Verification

There are three techniques currently being used for RISC-V processor verification: directed tests, constrained random test generation and test generation and execution.

A. *Directed Tests*

Directed test suites are an established technique, however, what has not been previously done is the measurement of instruction coverage of these test suites. Also, with the vector extensions to the RISC-V ISA, the difficulty involved in building a comprehensive test suite is increased exponentially. The vector instructions have 90 different configurations, and nearly 100 instructions. This is obviously a complex problem.

With this paper, the authors report new instruction coverage metrics, with the coverage tool included in the ISS. An example of the coverage results for the RV32I compliance test suite is shown in Fig. 2.

The authors are also developing a directed test suite ("Vector Test Suite") for the RISC-V vector instructions. Data on this test suite will be available later this year.

B. *Constrained Random Test Generation*

Constrained random test generation for SoC DV is also an established technique. However, for processor DV, this needs to be an Instruction Stream Generator (ISG). Google has developed and made open source an ISG for RISC-V [5].

Fig. 3 shows the basic flow for RISC-V processor DV using the ISG. This flow was originally developed to use a trace or signature compare methodology, but is now being evolved to support a step-and-compare methodology using the ISS

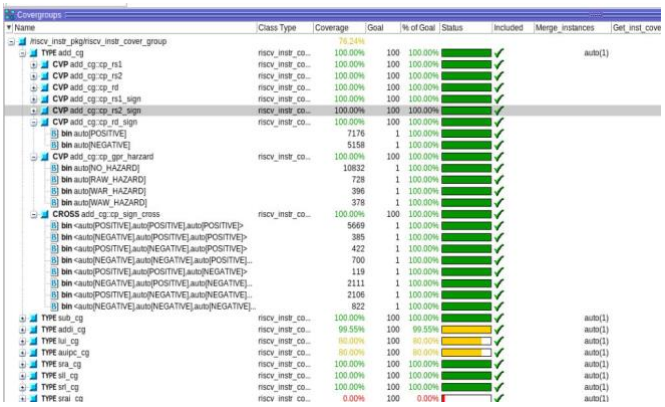

Fig. 2. Example of instruction coverage results for the RV32I compliance test suite.

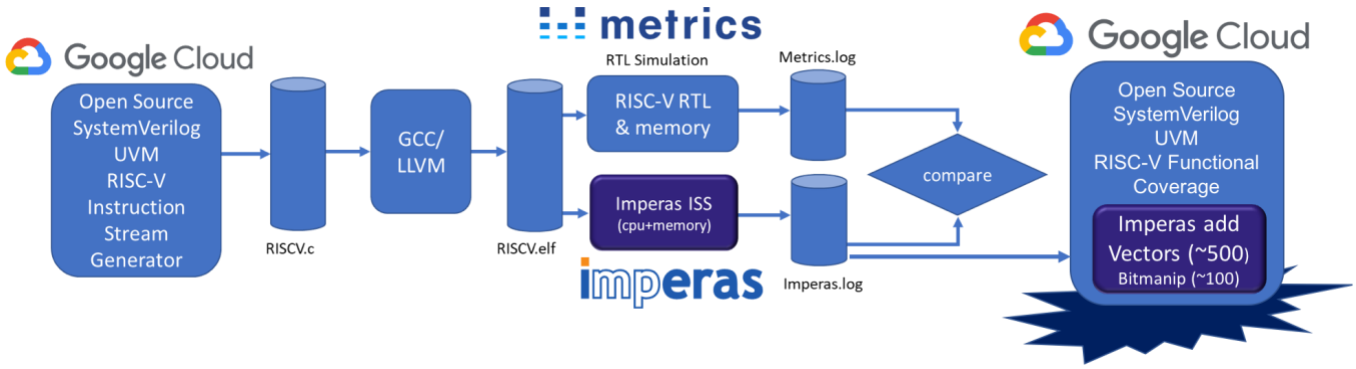

Fig. 3. RISC-V processor DV flow with Google open source ISG and Imperas ISS as reference model. The Metrics cloud simulation environment for SystemVerilog is shown in this diagram, however, any UVM-compliant SystemVerilog simulator could be used.

encapsulated in SystemVerilog, as shown in Fig. 4. Fig. 5 shows the step-and-compare flow.

This flow has been implemented for the testing of the Ibex core that was originally developed by ETH Zurich under the name "Zero-riscy" [6]. Recently this was adopted by LowRISC as Ibex [7]. Ibex implements the RISC-V RV32IMC instructions, which is the 32-bit RISC-V processor with integer (I), multiplier/divider (M) and compressed (C) instructions.

Table 1 shows the different categories of bugs found in the Ibex processor using this approach, while Fig. 6 shows an example of two types of bugs found.

Table 1. Categories of bugs found using the ISG-based DV flow.

Bug Category	Percentage of Bugs Found
Debug mode	31.3%
Illegal/hint instructions	25.0%
Interrupt	18.8%
Memory access fault	12.5%
Pipeline issue	6.3%
Others	6.3%

C. *Test Generation and Execution*

An alternative, and complementary approach to test generation for processor DV, which also can be applied to SoC DV, is the generation of tests as an executable which can be run on the RTL. In this approach, tests are randomly generated, then run on the processor reference model. The results from running the tests on the processor reference model are then combined with the random tests and used as reference test results. This flow is shown in Fig. 7 [8].

V. PE, MUTLIPLE PE AND NOC-PE VERIFICATION

The key pieces of SoC DV include verification of single PEs, verification of multiple PEs and verification of the interface between the PE and the Network on Chip (NoC).

A. *Verification of Processing Elements*

In many RISC-V based SoCs targeted at AI applications, the architecture includes Processing Elements (PEs) which have more than one RISC-V processor, plus an AI accelerator, plus some custom logic. The custom logic is typically comprised of custom instructions added to the RISC-V processors, plus additional logic for controlling the communications between processors. For these PEs, the individual processors (including custom instructions) must be verified as discussed above. However, with the integration of multiple processors into the PE, there can be different interactions that cannot be tested at the individual processor level.

For this level of integration, the PE can also be modeled using the same instruction accurate techniques that were used to model individual processors. The OVP APIs are used to build a model

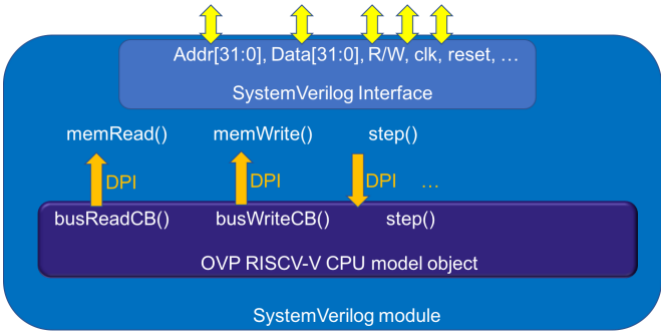

Fig. 4. Block diagram of SystemVerilog encapsulation of the RISC-V processor model.

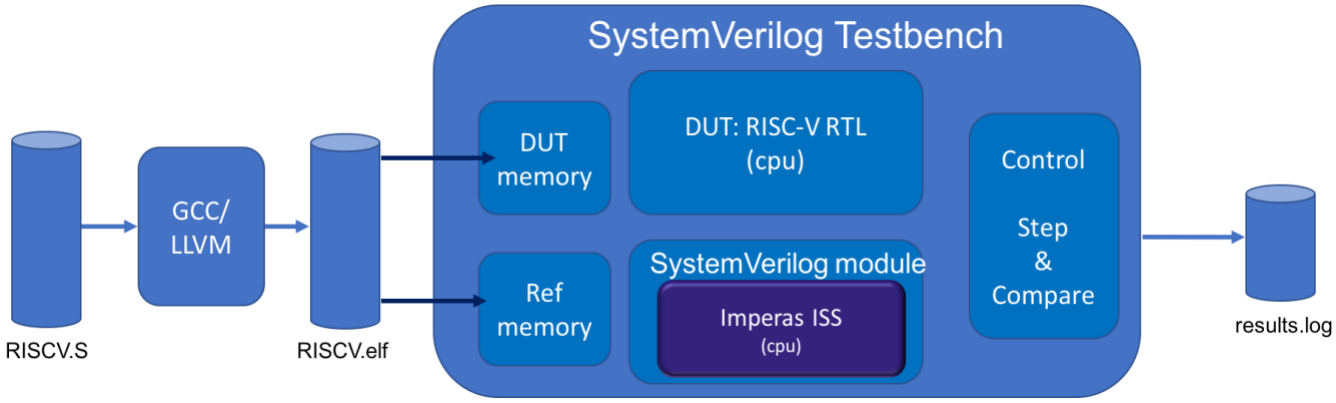

Fig. 5. Encapsulated ISS in the SystemVerilog testbench for a step-and-compare DV flow.

of the PE, and the same SystemVerilog encapsulation techniques are used to encapsulate the model of the PE, again enabling step-and-compare verification of the PE RTL.

A key piece here is the test generation. Certainly constrained random test generation could work, however, this might spend too many cycles "re-verifying" the individual processors and not focusing on the new, unique interactions at the PE level of integration. Another possibility is to run actual software meant to execute on the real PEs. This should bring out these additional interactions.

In these AI architectures, often one PE interacts regularly with multiple neighbor PEs. It is unclear how best to verify these PE-PE interactions. Two ideas being explored now are 1) to combine the instruction accurate models with RTL simulation; and 2) to combine the instruction accurate models with hardware emulation. In the first scenario, one might have one PE represented in RTL, and the remainder of the PEs as IA models. This could enable the IA models to run the actual software, generating more interesting "stimuli" for testing the RTL PE. In the second scenario, something similar to the first is contemplated, however, in this situation the RTL blocks would be implemented in the hardware emulator. Such a hybrid IA simulation-emulation environment is shown in Fig. 7 [9].

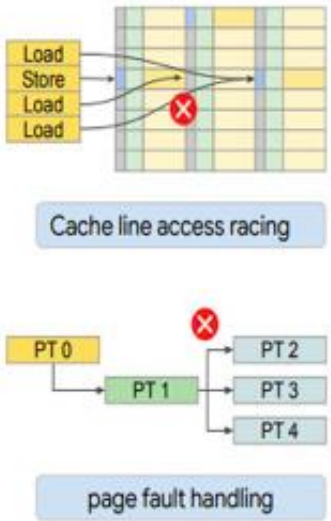

Fig. 6. Examples of two types of bugs found with the ISG-based flow.

B. *Processor/PE—NoC Verification*

Verification of the interface between a processor or processor subsystem and a NoC is a well-established process. This element of RISC-V verification is raised because there has been only a limited number of RISC-V based SoCs built using the various NoCs, so DV engineers should realize that this is not the fully mature NoC interface that one receives when other processor architectures are used.

VI. Conclusions

The RISC-V instruction set architecture is capturing the attention and interest of many companies because of its openness. However, the relative lack of maturity of the RISC-V processor IP means that verification of the RISC-V RTL is a critical task for SoC development. Also, with many RISC-V SoCs, additional DV needs to be done at higher levels of integration on the SoC. Together, these DV challenges require new tools and flows to meet the requirements of high quality SoCs.

Acknowledgment

The authors wish to thank Richard Ho and Tao Liu of Google LLC for their help with the Google ISG.

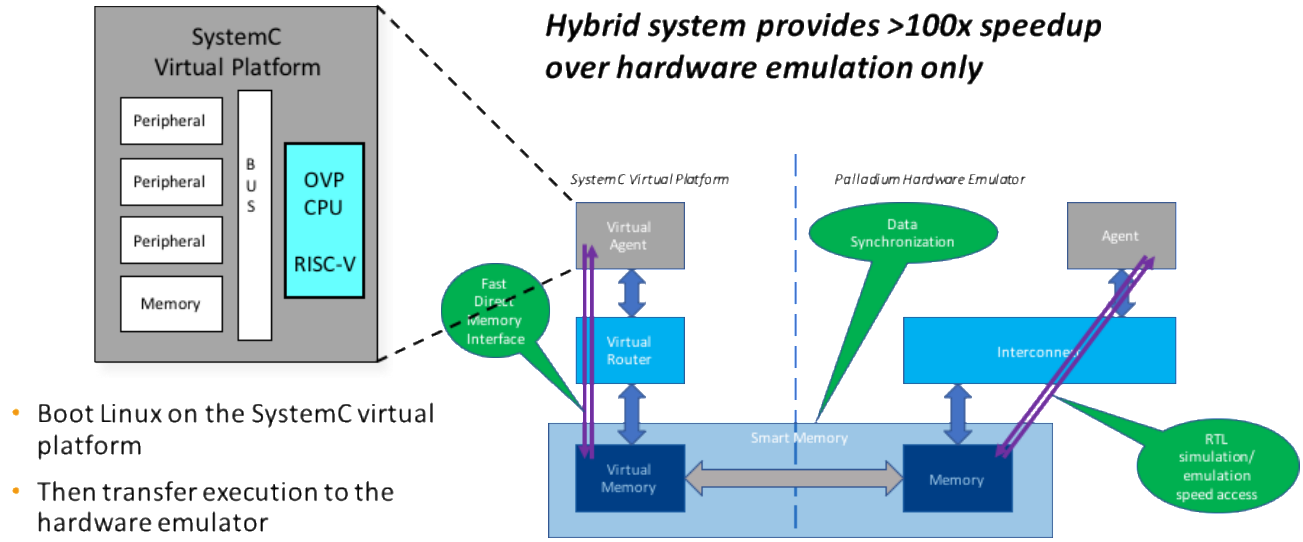

Fig. 7. Block diagram of a hybrid IA simulation-hardware emulation environment [9].

REFERENCES

[1] RISC-V Foundation ISA specification at https://riscv.org/specifications/

[2] RISC-V Foundation Compliance GitHub repository: https://github.com/riscv/riscv-compliance

[3] OVP (Open Virtual Platforms) http://www.ovpworld.org

[4] L. Moore, S. Davidmann, L. Lapides, "Methodology for Implementation of Custom Instructions in the RISC-V Architecture," Embedded World 2019, available at http://www.imperas.com/ew19-paper-on-methodology-for-implementation-of-custom-instructions-in-the-risc-v-architecture

[5] Google ISG GitHub repository at https://github.com/google/riscv-dv

[6] P. D. Schiavone et al. "Slow and steady wins the race? A comparison of ultra-low-power RISC-V cores for Internet-of-Things applications." 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS 2017)

[7] Ibex by LowRISC https://github.com/lowRISC/ibex

[8] S. R. Choudhury, Shajid T., J. John, George S., "Verifying RISC-V Vector and Bit Manipulation Extensions using STING Design Verification Tool," RISC-V Summit 2019, https://content.riscv.org/wp-content/uploads/2019/12/12.11-16.40b-Verifying-RISC-V-Vector-and-Bit-Manipulation-Extensions-using-STING-Design-Verification-Tool.pdf

[9] K. McDermott, L. Lapides, "Fast Processor Models for Software Bring-Up and Hardware-Software Co-Verification," CDNLive EMEA 2019, paper SVG03, https://www.cadence.com/en_US/home/cdnlive/emea-2019/proceedings.html