2020 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

A Comprehensive Verification Platform for RISC-V based Processors

Emre Karabulut, Design and Verification Engineer, Yonga Technology Microelectronics R&D, Istanbul, Turkey (*emre.karabulut@yongatek.com*)

Berk Kisinbay, Design and Verification Engineer, Yonga Technology Microelectronics R&D, Istanbul, Turkey (*berk.kisinbay@yongatek.com*)

Abdullah Yildiz, Design and Verification Engineer, Yonga Technology Microelectronics R&D, Istanbul, Turkey (*abdullah.yildiz@yongatek.com*)

Rifat Demircioglu, Design and Verification Engineer, Yonga Technology Microelectronics R&D, Istanbul, Turkey (*rifat.demircioglu@yongatek.com*)

***Abstract*—Although there has been growing demand for RISC-V based verification solutions due to the outstanding performance and the trend of RISC-V based SoCs, there is no standard verification method for the RISC-V based SoCs. This paper presents a comprehensive UVM based verification platform for this purpose. The platform provides prosperous test vectors with standard test suites and custom test sets. The wide range of test vectors provide system level and block level verification of all RISC-V based SoCs. Besides, the platform has functional simulators and robust tests for branch predictor and cache blocks, which are the two important components of any modern processor today. Another contribution is an automated verification environment that is integrated with other environments and test suites. Also, the verification environment enriches itself with code, functional and instruction coverage metrics. The verification platform can support all RISC-V standard ISA and extensions. We evaluate our verification platform on a customized RISCV-BOOM based SoC and obtained 81% functional coverage, 80% code coverage, and 82% instruction coverage.**

Keywords—RISC-V; verification; simulation; test; cache; branch predictor; SoC

I. INTRODUCTION

RISC-V has been enabling SoC designers to benefit from a wide ecosystem, which includes more than 100 implementations along with different instruction set simulators, toolchains, etc. [1]. Although RISC-V's standard ISA has been designed for general-purpose computing, it has also lowered the barrier to design for customized computing and domain specific computing by providing specialized instruction set extensions [2, 3]. In addition, RISC-V's free and open license model has lead RISC-V to become more popular on a global scale. Recent research shows that there will be about 62 billion RISC-V based CPU cores in the market by 2025 [4,5].

RISC-V allows to reduce non-recurring engineering (NRE) costs by open-source strategy, but it brings verification and integration challenges [6]. RISC-V ISA does not set a condition or restriction on RISC-V CPU design, and this could be a significant problem when implementing a modern CPU design that includes multi-stage pipeline, multi-level caches, out-of-order execution, etc. For this reason, each RISC-V CPU design requires rigorous verification to ensure that it meets the ISA standards that are defined by RISC-V foundation. Another challenge that engineers are confronted with is the process of integration of the CPU with other IPs and blocks in a system. Considering all these factors, verification of a RISC-V based CPU should not only include the verification of the core itself, but rather the verification of the whole system.

In this paper, we present a comprehensive RISC-V verification platform. Our verification platform supports all extensions of RISC-V ISA and can be tailored to verify any RISC-V based system. We also provide simulators and targeted test cases for cache and branch predictor blocks that enable us to verify these components at block level. Last but not the least, we integrate existing RISC-V test environments within our verification flow to provide a complete RISC-V verification platform.

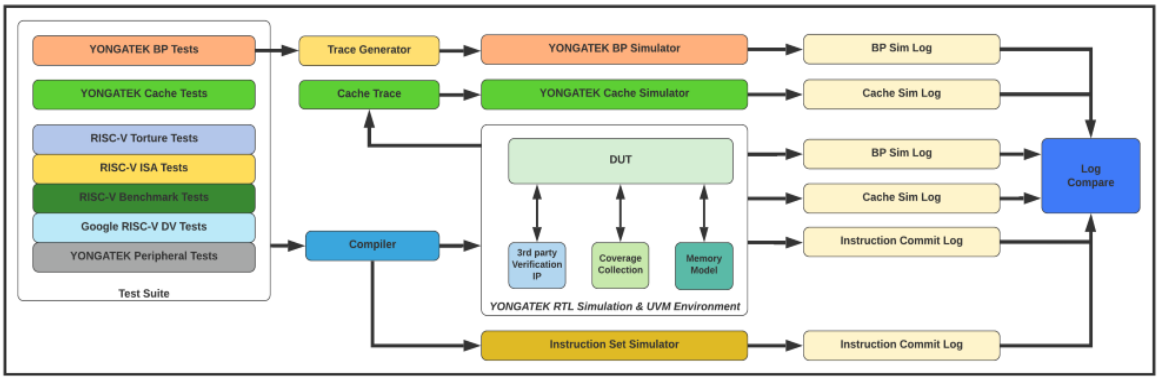


Figure 1. YONGATEK RISC-V Verification Flow

II. Related Work

Although RISC-V ISA based microprocessors are promising candidates for low-cost, area-optimized, and power-efficient ISA, there are a limited number of works to support and verify low to high-performance RISC-V cores. We can group existing RISC-V verification environments based on two different techniques, namely functional and formal verification. In functional verification, usually a test set [7-9] in the form of assembly programs is run on processor under verification (PUV). The output that is obtained from the PUV is either instruction commit log or a signature that is unique to a running test. Each output is then compared with the output that is obtained from a trusted golden model. The golden model is usually an instruction set simulator (ISS) such as Spike [10]. Google RISC-V DV [11], uGP [12, 13] can be demonstrated as examples of core-level functional verification environments. Some environments also provide system and block-level functional verification flows. Authors in [14] propose a SystemC based platform which allows verification of single and multi-core systems and supports RV32/64 IMAC ISAs. Authors in [15] present a configurable and hierarchical block-level verification methodology in which the system can be verified either at SoC or unit level, however, unit (i.e., core, pipeline, cache, branch predictor) and SoC-level verification are performed within different testbenches. In addition to that, the proposed environment is limited to single-issue, in-order pipeline microarchitectures and includes only block-level verification of L2 caches. Authors in [16] provides a UVM based block and chip-level verification environment for a customized RV32I ISA based microcontroller. Another work [17] proposes verification of core, cache, and memory by using a custom golden model of the system, which was developed by the authors itself.

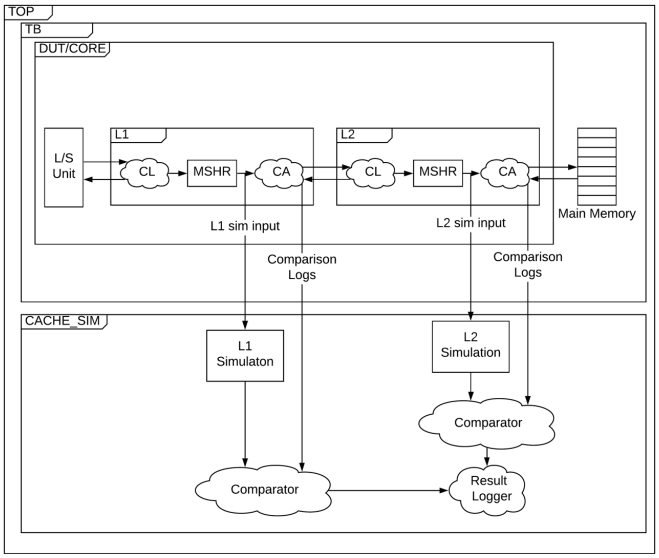
On the other hand, some formal verification techniques use mathematical models such as induction, bounded model checking, etc. to prove the correctness of the PUV. RISC-V Formal [18] is an open-source formal verification framework and provides formal descriptions of RISC-V ISA as well as a formal interface that enables verification of a RISC-V core in RTL simulation. MicroTESK [19] and Kami [20] is another formal verification environment in which formal methods are used to generate test programs.

There is no existence of a verification environment in the literature that can support all RISC-V ISA families within a unified platform by providing system-level verification. In addition, none of the previous works consider the verification of complex components such as cache and branch predictor via robust tests that are specific for each component. These blocks are more likely to contain bugs [21] which are hard to detect with instruction level verification. For this reason, they need to be tested more rigorously in order to cover corner cases and detect potential hazards.

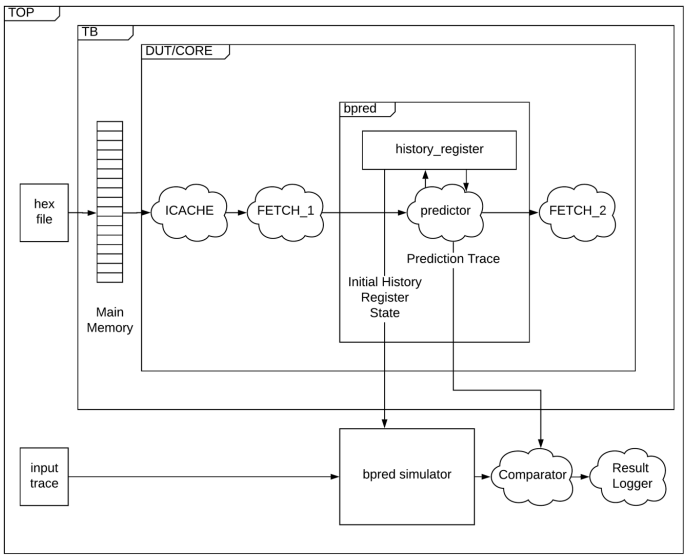
III. Proposed Methodology

A. Overview

In this work, we utilize and customize three different tools and integrate them into one flow (see Figure 1). The first one is riscv-dv [11]. It takes advantage of System Verilog and UVM for constrained randomness and re-



a.



b.

Figure 2. Flow diagrams for cache verification (a) and branch predictor verification (b) of the proposed verification platform

usability. The second one is riscv-tests [7], this tool includes fixed instruction tests and benchmarks. The third one is riscv-torture [9] similar-to riscv-dv but is uses Scala as its base. These are test generator platforms to generate large scale instruction streams. Although the platforms do not provide a verification environment, they allow to present a test generation methodology for out-of-order (OoO) cores as well as in-order cores. By following the test generation methodology, there is a work that uses ISS and one of the test generation platforms to verify a RISC-V core [8, 22]. The verification environment verification idea works basically by comparing the ISS's and the target core's committed instructions.

Although the golden model ISS and the target core should give the same output in overall, those verification methods cannot verify the internal stages. For instance, a branch predictor can operate faulty. Then the committed branch instruction might be correct even though the correction brings a miss-prediction penalty. As another example, any level of cache can faultily perform replacement policy, while the wrong replacement does not affect the committed instructions except for a miss-penalty. We propose a verification method that includes all advantages of the aforementioned verification methods and additional tests and block-level simulation models such as branch predictor and cache simulator.

In this flow, there are three sub-stages: test generation, simulation-background, and log comparison. We have combined riscv-dv, riscv-tests and riscv-torture with our flow (see Figure 1). The flow generates test cases with the test generation platforms. We also use benchmarks, such as Spec95 [23], as new test cases to enrich riscv-dv platform. Our test generation platform employs constrained randomness of UVM, while follows an improvement with benchmark level test cases. Moreover, we also add some characteristics to riscv-tests platform for creating custom test cases.

The simulation-background phase contains cache, branch, and ISS simulation stages. We create a testbench that does not produce only committed instructions, but also prints enriched reports, comparison logs, for our branch predictor and cache simulator models. The branch predictor and cache simulator models need input traces to model the target blocks in the core. Hence, the testbench also produces input traces obtained from different pipeline stages in the target core.

We create an automation system for the log comparison. The system compares two different committed instructions, one of them is from Spike ISS and the other one is from the target core. For the block level log comparison, the automation system compares the branch predictor and cache simulator models' logs. A cache test and a branch prediction pass when all the required logs match (see Figure 3, 4).

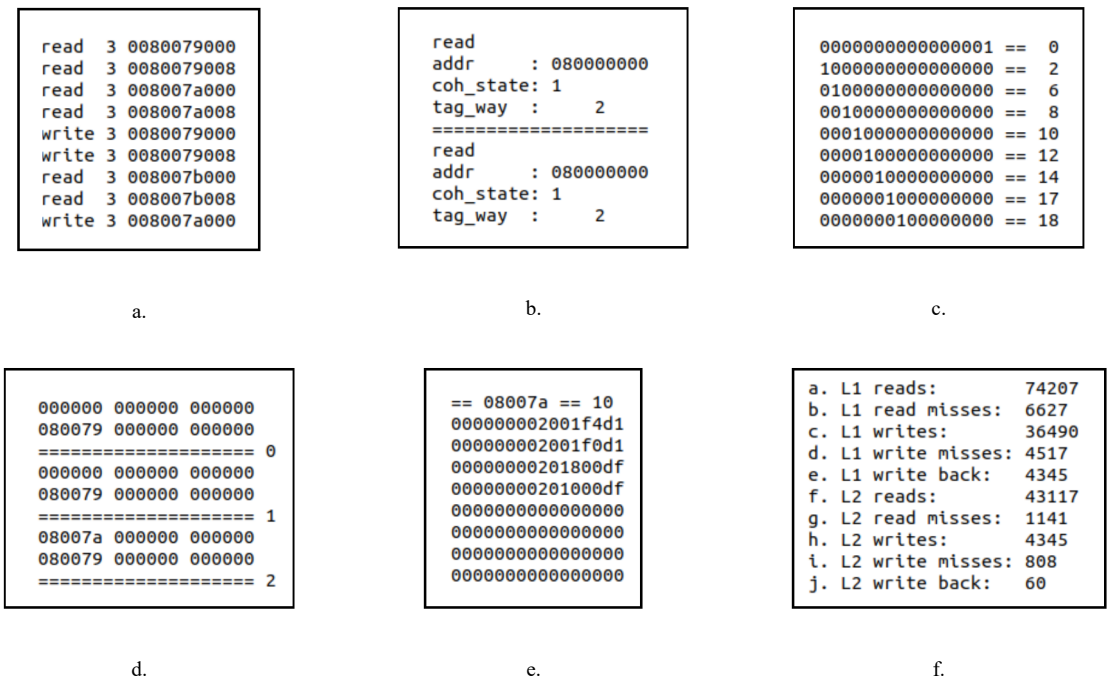


Figure 3. Cache simulator's input trace (a) and outputs: run log (b-e) and result report (f)

B. Cache Simulator

Our cache simulator has four key features. First, it provides run-time and compile-time flexibility that support different cache structures and parameters, e.g. blocking/non-blocking direct-mapped, set-associative, fully associative cache. Second, the system supports load-store and atomic instructions. Third, our proposed method does not just enable to test the structure functionality, but it also gives a performance analysis report that allows to design parameters updates in the behavioral simulation step in the design flow. Fourth and last, the simulation model presents a comprehensive and prosperous simulation output log for comparison.

Figure 2-a shows the cache simulator environment and possible I/O of the out-of-order target hardware. The hardware has a non-blocking, set-associative cache structure. The target cache structure follows the pseudo

replacement policy, while the requests must pass through Miss Status Holding Registers (MHSRs) due to the out-of-order behavioral of the given hardware.

The proposed cache simulator requires following parameters for its own configuration: number of the cache levels, the capacity of the cache levels, one block line size, the associativity of cache levels and the replacement policy. After the configuration step, the simulation model requires an input trace that is the same trace format given to the target hardware (see Figure 3-a). The simulation model gives a detailed Hit/Miss log for the corresponding memory operation report (see Figure 3-b). If there is a mismatch between the hardware detailed report and the simulation model, the platform provides also supportive output logs such as replacement register content (Figure 3-c) and tag-array content (Figure 3-d) for debugging. To complete the full check mechanism, the simulation model also compares the final memory contents of any cache level and the main memory. In this way, our proposed method checks each transaction of the target hardware control units, while it also verifies the memory model against any data loss case in the content.

Our presented simulation method is written in C/C++ to provide outstanding performance. The log comparison automation system is written as Python scripts. Figure 3-f illustrates how the simulation system provides a performance analysis report. In other words, the simulation model can run stand-alone. Hence, the simulation model can provide a performance analysis with benchmark inputs. Meanwhile, the system leads the designer to pick the optimal cache parameters.

C. Branch Predictor Simulator

Another major contribution is a branch predictor simulation model for the proposed RISC-V verification environment. Figure 2-b shows the branch predictor simulator environment and possible I/O of the out-of-order target hardware. Like the cache simulator model, the branch prediction verification platform is also configurable with different structures and parameters e.g. Bimodal, G-Share, Hybrid predictor models. The simulation model can run as stand-alone for performance analyze, while it also performs to verify the branch predictor corner case for robust test cases such as nested conditional branches.

The configuration parameters can provide run-time and compile-time flexibility. The simulation platform also reuses the cache simulator components to support the set-associative or direct-map branch target buffer (BTB). Thus, the configuration parameters should contain the index size for the bimodal table and the global history register bit width. The hybrid predictor model requires both bimodal and the G-Share parameters. For the BTB, the model

```
pc      : 008000106c
taken   : 1
commit history : 0xffc
====================        163
pc      : 0080001310
taken   : 1
commit history : 0xff9
====================        164
pc      : 008000143c
taken   : 1
commit history : 0xff3
====================        165
pc      : 008000106c
taken   : 1
commit history : 0xfe7
====================        166
```

a.

```
number of predictions: 2822
number of mispredictions: 57
misprediction rate: 2.02%
FINAL GSHARE CONTENTS
0 0xaa
1 0xaa
2 0xaa
3 0xaa
4 0xaa
5 0xaa
6 0xaa
7 0xaa
8 0xae
9 0xaa
10 0xaa
11 0xaa
```

b.

Figure 4. Branch predictor simulator's outputs: run log (a) and result report (b)

needs the associativity of the BTB and the size of the BTB. By contrast, the BTB model can be disabled or enabled by depending on the requirement. The branch predictor tests provide the same input for both the target hardware and the simulator.

Figure 4-a presents the related work's output log for the G-Share configuration. The validation platform expects to match each line printed by the model with the hardware output. Moreover, the validation model completes a comparison of the final content of the predictor tables of the simulation model and the hardware. Figure 4-b proposes the performance analysis report for the G-Share configuration. According to the reports, the parameters can be changed by a designer team if it is required.

D. Functional/Code Coverage Metrics

Our other contribution is that we collect code coverage and functional coverage with our own test plan. We have created about 10,000 covergroup elements, containing coverpoints and crosses. All functional coverage elements are reusable and able to be enabled and disabled, hence the verification environment is eligible to be used for different RISC-V cores. We created the coverage elements for the requirement specifications of the customized RISCV-BOOM [24].

Additionally, we also created a coverage matrix for the instruction families. The instruction families are all 64-32 I-M-A-F-D instructions. Figure 5 shows a piece from our coverage matrix. To cover all instruction, we use test cases already implemented with prior works and also our extra test cases. In the next section, we provide more detailed information for our coverage results and test cases.

IV. PRELIMINARY RESULTS

We notify that due to such variance in simulation tool technology and use of different verification tools (or versions) leads a comparison to serve as a first order estimate rather than an idealized method. Despite this fact, our verification platform outperforms earlier work in flexibility, coverage, comprehensiveness.

We used 333 test cases in our verification environment. Figure 5 shows the coverage results for a customized SoC, based on RISC-V BOOM [24] core. We collected up to %100 coverage for 64 I-M-A-F-D family instruction opcodes with the 333 test cases. In terms of code coverage, we achieved 79.5% statement and 75.54% branch coverage. Also, Table I illustrates that we achieved up to %100 instruction coverage for different instruction types. Although each instruction can have opcode, immediate, etc. fields in accordance with instruction type, we provide corresponding fields (register fields 1, 2, 3, and 4) instead of entire fields to show average coverage results in one compact table. For the AMO instructions' coverages are relatively low compare to the other instruction types because our AMO instruction test cases aim to verify the internal memory blocks functionality rather than decoding stages.

Our branch and cache simulation models demonstrated significant performance, as expected. Our test results show that there are at least tens of mistakes that can be caught by our simulator models but not the instruction level. For example, the target architecture does not decrease or increase the two-bits counter values, as expected. Thus,

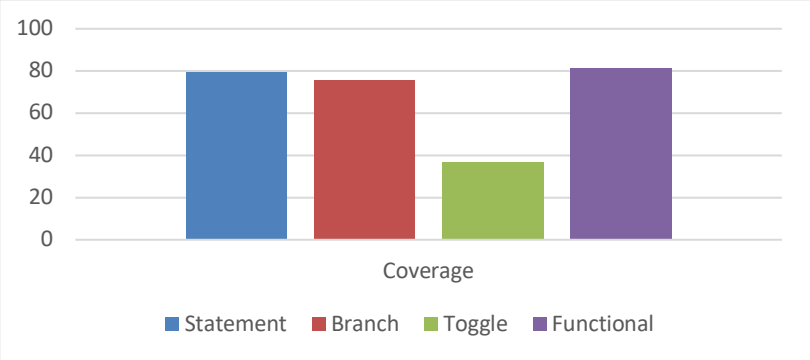


Figure 5. Code and functional coverage results

the misprediction penalty is heavier than expected. However, the error cannot be caught with other verification environments due to comparing just the committed instructions. Our branch predictor caught the error with detailed log contents. Also, our cache simulator detected wrong order write-back execution. This mistake was the end of the test; hence, the other test platform was not able to catch the error while our cache simulator demonstrated outstanding performance by detecting the group of errors.

Table II gives a comparison between our verification environment and other system-level verification environments with respect to five different criteria. The table shows that environment I [14], III [16], and IV [17] do not provide any robust tests and simulation models for block-level verification. As previously stated, environment II [15] only supports block-level verification of L2 cache and single-issue, in-order pipeline architectures with a limited set of robust tests.

Table I. Instruction coverage results

	Register Fields			
Instruction Type	***Field #1***	***Field #2***	***Field #3***	***Field #4***
LOAD	100.0	94.20	N/A	N/A
STORE	96.88	99.22	N/A	N/A
BRANCH	100.0	100.0	N/A	N/A
JALR	56.25	62.50	N/A	N/A
JAL	96.88	N/A	N/A	N/A
LOAD-FP	57.29	70.83	N/A	N/A
STORE-FP	46.88	68.75	N/A	N/A
OP	100.0	100.0	100.0	N/A
OP-32	100.0	100.0	100.0	N/A
OP-IMM	100.0	100.0	N/A	N/A
OP-IMM-32	100.0	100.0	N/A	N/A
OP-FP	94.06	92.50	90.56	N/A
AMO	12.07	4.38	13.07	N/A
(N)MADD/(N)MSUB	85.68	86.2	24.5	89.1

Table II. Comparison of RISC-V verification environments

	YONGATEK Env.	**Env. I [14]**	**Env. II [15]**	**Env. III [16]**	**Env. IV [17]**
Development environment	UVM	SystemC, TLM	Portable stimulus, UVM	UVM	Verilog RTL
Stimuli type	constrained rnd. tests directed tests ISA unit tests C benchmarks	ISA unit tests bare-metal tests OS-level tests	constrained rnd. tests directed tests	ISA unit tests	ISA unit tests C benchmarks
Submodule robust tests (cache, branch predictor, peripherals, etc.)	Yes	No	Yes	No	No
Peripheral interface tests	Yes	Yes	Yes	No	No
Block-level simulation models	Yes	No	Yes	No	No

V. CONCLUSION

This paper proposes a configurable and enlarged verification environment for RISC-V ISA family microprocessors. Our verification environment supports all target ISA. Our platform produces random test cases with Google RISC-V DV infrastructure, while we have our own robust test cases, our own log comparison

mechanism and simulation models for block-level verification. Our test environment gathers different features of different verification environments under a single platform. Although our verification environment is a promising candidate to support all RISC-V ISA and has simulation models for cache unit and branch predictor, future research can extend the proposed framework to have more simulation models for more detailed block-level verification and to have higher code coverage results.

ACKNOWLEDGMENT

We thank reviewers for their valuable feedbacks. This research is supported in part by Yonga Technology Microelectronics R&D. We acknowledge Synopsys for their VCS verification environment.

REFERENCES

[1] RISC-V International, https://riscv.org/ (Accessed: 1 February 2020).

[2] E. Alkim, H. Evkan, N. Lahr, R. Niederhagen, and R. Petri, "ISA extensions for finite field arithmetic," IACR Transactions on Cryptographic Hardware and Embedded Systems, 2020, 219-242.

[3] J.L. Hennessy, and D.A. Patterson, "A new golden age for computer architecture," Communications of the ACM, 2019, 62.2: 48-60.

[4] Semico Research (2019), "RISC-V market analysis: The new kid on the block," https://semico.com/sites/default/files/TOC_CC315-19.pdf (Accessed: 9 May 2020).

[5] RISC-V Foundation News (2019), "Semico forecasts strong growth for RISC-V," https://riscv.org/2019/11/9679/ (Accessed: 9 May 2020).

[6] T. Anderson, "Verification challenges for RISC-V adoption," https://www.gsaglobal.org/forums/verification-challenges-for-risc-v-adoption/ (Accessed: 9 May 2020).

[7] RISC-V Foundation, "RISC-V tests," https://github.com/riscv/riscv-tests (Accessed: 28 July 2020).

[8] RISC-V Compliance Task Group, "RISC-V compliance tests," https://github.com/riscv/riscv-compliance (Accessed: 28 July 2020).

[9] Y. Lee and H. Cook, "RISC-V torture tests," https://github.com/ucb-bar/riscv-torture (Accessed: 28 July 2020).

[10] RISC-V Foundation, "Spike RISC-V ISA simulator," https://github.com/riscv/riscv-isa-sim (Accessed: 28 July 2020).

[11] Google, "SV/UVM based instruction generator for RISC-V processor verification," https://github.com/google/riscv-dv (Accessed: 28 July 2020).

[12] P.D. Schiavone et al., "An open-source verification framework for open-source cores: A RISC-V case study," International Conference on Very Large Scale Integration (VLSI-SoC), IEEE, 2018, pp. 43–48.

[13] C. Duran et al., "Simulation and formal: The best of both domains for instruction set verification of RISC-V based processors," International Symposium on Circuits and Systems (ISCAS), IEEE, 2020, pp. 1-4.

[14] V. Herdt, D. Große, P. Pieper, and R. Drechsler, "RISC-V based virtual prototype: An extensible and configurable platform for the system-level," Journal of Systems Architecture, 2020, 101756.

[15] A. Chandra and M. Bartley, "A Hierarchical and Configurable Strategy to Verify RISC-V based SoCs," T&VS, 2020.

[16] R. Molina-Robles et al., "A compact functional verification flow for a RISC-V 32I based core," Conference on PhD Research in Microelectronics and Electronics in Latin America (PRIME-LA), IEEE, 2020, pp. 1-4.

[17] A. Oleksiak, S. Cieślak, K. Marcinek, and W.A. Pleskacz, "Design and verification environment for RISC-V processor cores," Mixed Design of Integrated Circuits and Systems, IEEE, 2019, pp. 206-209.

[18] Symbiotic EDA, "RISC-V formal verification framework," https://github.com/SymbioticEDA/riscv-formal (Accessed: 1 July 2020).

[19] M. Chupilko, A. Kamkin, and A. Protsenko, "Open-source validation suite for RISC-V", International Workshop on Microprocessor/SoC Test, Security and Verification (MTV), IEEE, 2019, pp. 7–12.

[20] J. Choi, M. Vijayaraghavan, B. Sherman, A. Chlipala, Arvind, "Kami: A platform for high-level parametric hardware specification and its modular verification," Proceedings of the ACM on Programming Languages, ACM, 2017, pp. 1-30.

[21] P. Kocher et al., "Spectre attacks: exploiting speculative execution," Symposium on Security and Privacy (SP), IEEE, 2019, pp. 1-19.

[22] L. Moore, R. Ho, D. Letcher, K. McDermott, "RISC-V compliance & verification techniques processor cores and custom extensions", DVCon Europe, 2019.

[23] Standard Performance Evaluation Corporation, "SPEC CPU95 Benchmarks," https://www.spec.org/cpu95 (Accessed: 10 April 2020).

[24] K. Asanovic, D. A. Patterson, and C. Celio, "The Berkeley out-of-order machine (BOOM): an industry-competitive, synthesizable, parameterized RISC-V processor," University of California at Berkeley, 2015.