COMPLETE COURSE OUTLINE

RISC-V Verification Training

CPU fundamentals, RISC-V CPU verification and software-driven RISC-V SoC verification.

Delivery	Live online, instructor-led
Structure	Three connected parts and 24 modules
Learning format	Teaching, examples, quizzes, practical exercises and review

PROGRAMME PURPOSE

Develop practical verification capability across CPU architecture, RISC-V CPU verification and SoC integration through structured teaching, open-source workflows and hands-on exercises.

Programme overview

As RISC-V adoption expands, verification teams need engineers who can reason across CPU architecture, UVM-based CPU verification and software-driven SoC integration. This live online programme develops those capabilities through structured teaching, quizzes, guided examples and practical exercises.

The course is designed for engineers who are new to CPUs or CPU verification, including engineers verifying the integration of a CPU into a System-on-Chip (SoC). Participants progress from CPU and RISC-V fundamentals to CPU-level and SoC-level verification workflows that can be related to real engineering projects.

Course options

Pathway	Coverage
Part 1	CPU and RISC-V Basics - CPU architecture, verification strategies, instruction-stream generation, functional coverage and riscv-dv practice.
Part 2	RISC-V CPU Verification - RISC-V ISA concepts, software toolchains, RARS, open-source UVM and riscv-dv workflows, architectural compliance and CPU microarchitecture.
Part 3	RISC-V SoC Verification - software-driven SoC verification, self-checking tests, assertions, coverage, debug and signoff.

Engineers who already understand CPU verification but are new to RISC-V may be able to follow a focused RISC-V CPU verification pathway. Alpinum will confirm the appropriate scope against the participant's or team's experience and learning objectives.

Learning outcomes

By the end of the relevant programme scope, participants should be able to:

- Describe current best-practice CPU and CPU-based SoC design verification strategies.
- Explain the principal methodologies, tools and languages used in CPU and CPU-based SoC verification.
- Apply those methodologies, tools and languages to a basic RISC-V CPU design and an SoC design based on RISC-V.
- Analyse a real RISC-V CPU or SoC and propose an appropriate design verification strategy.
- Discuss CPU and CPU-based SoC design verification topics confidently with engineering colleagues.
- Contribute more effectively to RISC-V CPU and SoC verification projects.

Who should attend

- Graduate recruits and university placement students.
- Design and verification engineers who want to understand CPU and CPU-based SoC verification.
- Engineers moving into CPU verification, RISC-V verification or CPU integration.
- Engineering managers who need a working understanding of CPU and SoC verification strategies.

Prerequisites

Some experience with design verification and SystemVerilog using the Universal Verification Methodology (UVM) is beneficial. Contact Alpinum if you are unsure whether the complete programme or a focused pathway is the best fit.

How the training works

Instructor-led teaching: Live online sessions establish the architecture, methodology and verification flow.

Guided examples: Participants see workflows demonstrated before applying them independently.

Practical exercises: Exercises cover code generation, simulation, coverage, compliance, software-driven SoC testing, debug and signoff.

Review and feedback: Joint classroom sessions help participants consolidate techniques and relate the exercises to real project work.

Part 1: CPU and RISC-V Basics

PART FOCUS

Establish the CPU architecture and verification foundation before applying it through functional coverage and riscv-dv exercises.

1. Introduction to CPU

Approximately 1 hour of teaching

1.1 CPU architectures and Instruction Set Architectures

- What an Instruction Set Architecture (ISA) is and why ISAs are needed for code and binary portability.
- Introduction to CPU architectures.

1.2 CPU microarchitectures

- A simple CPU block diagram.
- Basic operation of a simple CPU microarchitecture.

2. How do we verify a CPU?

Approximately 2 hours of teaching

2.1 Overall CPU verification strategies

- Compare different approaches to CPU verification and their relative advantages and disadvantages.
- Explain the hierarchical, simulation-based verification approach used during the course.

2.2 Unit-level verification

- How standard SystemVerilog and UVM testbenches can be used for unit-level verification.
- Examples covering fetch and branch prediction, decode and dispatch, integer execution, load/store execution, floating-point execution, instruction caches, data caches, Memory Management Units (MMUs) and Control and Status Registers (CSRs).
- More specialised units may include vector units and reorder buffers.
- Detailed unit-level verification is specific to each design and is therefore outside the scope of this course.

2.3 CPU-level verification

- How binary machine code can be used to verify a CPU.
- How assembler code can be used for CPU verification.
- How instruction-stream generation can be used to produce assembler tests.
- Example of a UVM testbench that runs generated assembler, converted into binary, on the CPU.

2.4 System-level integration

- Introduction to the objectives and techniques used to verify the integration of a CPU into an SoC.
- CPU integration verification is covered in more detail later in the programme.

3. Basic CPU-level verification and tooling

2 hours of teaching plus examples

3.1 Instruction-stream generators

- How instruction-stream generation works.
- Why constrained pseudo-random generation is needed.
- The types of constraints required for CPU verification.

3.2 Open-source riscv-dv tooling

- Overview of riscv-dv as an example of an instruction-stream generator.

- Examples of running riscv-dv.
- riscv-dv is covered in greater detail later in the programme.

3.3 Impact of adding custom instructions

- How adding custom instructions can affect the verification environment and process.
- This topic is explained at a high level because implementation details are specific to each design.

4. CPU microarchitectures for faster code execution

4 hours of teaching

4.1 Reviewing different microarchitecture styles

- Review and comparison of different microarchitecture styles.
- Examples include RISC, CISC and VLIW architectures.
- These architectural approaches are explored in greater detail later in the programme.

5. Verifying CPU microarchitectures

4 hours of teaching plus examples

5.1 Building functional coverage models for a microarchitecture

- Identify the microarchitecture features that need to be verified.
- Define suitable functional coverage points.
- Extend an existing functional coverage model.
- Run tests and review the resulting coverage.

5.2 Generating tests to hit functional coverage points

- Review the existing constraints.
- Extend the constraints to hit coverage points in the expanded functional coverage model.
- Run targeted tests and review their contribution to coverage closure.

6. riscv-dv practical exercises

At least 8 hours of independent study plus at least 1 hour of classroom review

Practical work

- Extend the functional coverage model and constraints file.
- Run tests and work towards coverage closure.
- Complete the exercises independently with access to support and feedback.

6.1 Joint classroom review and feedback

- Jointly review the completed exercises.
- Discuss the results, challenges and possible improvements.

Part 2: RISC-V CPU Verification

PART FOCUS

Apply the foundation to RISC-V software flows, open-source verification tooling, architectural compliance and advanced CPU microarchitecture.

7. Introduction to RISC-V

Approximately 2 hours of teaching

7.1 RISC-V ISA overview

- Understanding RISC-V instruction formats.
- RISC-V ISA modularity.
- RISC-V privilege modes.
- Introduction to the RISC-V memory model.

7.2 Assembly language for RISC-V

- Introduction to RISC-V assembly language.
- Review of example assembler programs.

8. Writing and running RISC-V programs

Approximately 1 hour of teaching plus examples

8.1 RISC-V software toolchains

- Overview of RISC-V software toolchains.
- Flow diagrams illustrating the software toolchain.
- The process for creating binary files from assembler and C code.

8.2 RISC-V Assembler and Runtime Simulator

- Introduction to the RISC-V Assembler and Runtime Simulator (RARS).
- Write and run example assembler programs for the basic RISC-V ISA.
- Run the example programs in RARS.

8.3 Converting C code into RISC-V assembler

- Write and run example C programs for the basic ISA.
- Convert C code into RISC-V assembler.
- Run the programs using RARS.
- Complete practical exercises involving assembler and C programs.

9. RISC-V practical exercises

At least 4 hours of independent study plus at least 1 hour of classroom review

Practical work

- Write and run code for RISC-V.
- Run RISC-V assembler and C programs using RARS.
- Complete the exercises independently with access to support and feedback.

9.1 Joint classroom review and feedback

- Jointly review the completed exercises.
- Discuss the results, difficulties and lessons learned.

10. Verifying CPU integration in an SoC

Approximately 2 hours of teaching

10.1 System-level integration verification

- Explain how software running on a mini-SoC can be used to verify CPU integration.
- Understand why higher-performance platforms may be needed to run some programs.
- Review examples of compliance suites and benchmarks.

11. Basic RISC-V verification and tooling

2 hours of teaching plus examples

11.1 Open-source riscv-dv tooling

- Detailed overview of riscv-dv.
- Examples of running riscv-dv.
- Review the generated code and how it is executed in simulation.
- Review pass and fail results.
- Review the resulting functional coverage.

11.2 Generating and running a first riscv-dv test

- Run riscv-dv.
- Simulate the generated output.
- Check the results and review the resulting functional coverage.

11.3 Reviewing the riscv-dv functional coverage model

- Review the functional coverage supplied by riscv-dv.
- Understand how the coverage model relates to verification objectives.

11.4 Reviewing the riscv-dv constraints model

- Understand how constraints are defined in riscv-dv.
- Modify selected constraints.
- Review the effect of those changes on generated tests and coverage.

12. riscv-dv practical exercises

At least 8 hours of independent study plus at least 1 hour of classroom review

Practical work

- Run the complete riscv-dv flow.
- Generate tests and simulate the output.
- Review results and functional coverage.
- Complete the exercises independently with access to support and feedback.

12.1 Joint classroom review and feedback

- Jointly review the completed riscv-dv exercises.
- Discuss the results and potential improvements.

13. Architectural compliance

2 hours of teaching plus examples

13.1 Understanding architectural compliance

- What architectural compliance means.
- Differences between RISC-V architectures and their compliance requirements.
- Introduction to architectural compliance suites.
- Detailed review of an example compliance suite.

13.2 RISC-V compliance suites

- Run an example RISC-V compliance suite.
- Review and interpret the results.

14. CPU microarchitectures for faster code execution

4 hours of teaching

14.1 Reviewing different microarchitecture styles

- Compare RISC, CISC and VLIW architectures.
- Understand why RISC architectures, and RISC-V specifically, are used.

14.2 RISC-V registers

- Overview of the RISC-V register set.
- Register addressing modes, including direct, indirect and immediate addressing.
- General-purpose and special-purpose registers.

14.3 Pipelining

- What pipelining is and which performance problems it addresses.
- Verification challenges introduced by pipelining.
- Techniques used to address these challenges, including out-of-order execution, branch prediction and register renaming.

14.4 Additional microarchitectures

- Superscalar processors and how they are designed.
- Multithreaded processors.
- Vector processors and Vector/SIMD instruction-set extensions.
- Multicore processors.

15. Final session

Course review

Review

- Review the principal RISC-V CPU verification concepts.
- Review the practical exercises.
- Discuss how the techniques can be applied to real verification projects.

16. Additional material

Supplementary content

Formal verification

- Introduction to how formal verification can support RISC-V CPU verification.

Part 3: RISC-V SoC Verification

PART FOCUS

Verify a RISC-V CPU within an SoC using software-driven testing, assertions, functional coverage, debug and signoff evidence.

17. Introduction to SoC verification

Approximately 2 hours of teaching

17.1 SoC verification

- The principal SoC verification tasks and challenges.
- Differences between IP-level and SoC-level verification.
- Metrics and signoff criteria used in SoC verification.

18. Introduction to the training SoC

Approximately 2 hours of teaching

18.1 The SoC used during the course

- Overview of the architecture of the training SoC.

18.2 The SoC verification environment

- Overview of the testbench.
- Tools and flows used to run tests and simulations.
- Adding checkers using assertions.
- Writing self-checking tests.
- Defining and implementing SoC functional coverage models.

19. SoC practical exercises

At least 4 hours of independent study plus at least 1 hour of classroom review

Practical work

- Review the environment used to run RISC-V code on the SoC.
- Understand the process for updating an existing test or writing a new test.
- Run existing tests.
- Update and run existing tests.
- Write and run new tests.
- Ensure that tests are self-checking.
- Add functional coverage.

19.1 Joint classroom review and feedback

- Jointly review the completed SoC exercises.
- Discuss the results and possible improvements.

20. SoC practical debug exercises

At least 2 hours of independent study, including classroom review

Debug workflow

- Understand the bug-finding, triage and debug workflow.
- Run a test on an SoC containing a known bug.
- Triage and debug the known bug.
- Review the debug process and findings.

21. How do we verify an SoC CPU?

Approximately 3 hours of teaching

21.1 Overall SoC verification strategies

- Compare different approaches to SoC verification and their relative advantages and disadvantages.
- Explain the SoC verification approach used during the course.

21.2 Verification of specific SoC features

- SoC integration verification.
- SoC reset and boot, including power-on sequence verification.
- SoC clock-control and clock-gating verification.
- SoC power-control verification.
- SoC interrupt verification.
- SoC system-control verification.
- SoC use-case verification.

22. SoC verification and signoff practical exercises

At least 4 hours of independent study plus at least 1 hour of classroom review

Practical work

- Write and run tests for a range of SoC features.
- Collect functional coverage and signoff metrics.
- Prepare an SoC signoff report.

22.1 Joint classroom review and feedback

- Jointly review the completed exercises.
- Discuss the collected evidence and signoff results.

23. Final session

Course review

Review

- Review the principal RISC-V SoC verification concepts.
- Review the practical exercises.
- Discuss how the techniques can be applied to real SoC verification projects.

24. Additional material

Supplementary content

Formal verification

- Additional guidance on how formal verification can support RISC-V SoC verification.

Discuss the right programme scope

NEXT STEP

Contact Alpinum with your team's experience, project context and learning objectives. We will help identify the most appropriate programme or focused pathway.

https://alpinumconsulting.com/services/training/riscv-verification-training/