RISC-V Verification Training Course

This course introduces participants to best-practice CPU Design Verification (DV) strategies so they can contribute effectively to real projects (see the objectives). It is aimed at engineers new to CPU and/or CPU verification, including verifying integration of the CPU into an SoC (System-on-Chip). or just understand more about it (target audience). It is delivered fully online with a large number of quizzes and practical exercises to practice the content covered in the lectures. The expected pre-requisites are shown here.

The course is split into 3 parts with links to further details:

1. Introduction to CPU and CPU verification (details)
2. RISC-V CPU verification (details) which focuses on how to verify a RISC-V CPU. It uses an open source UVM test bench and students are expected to use that for the practical exercises.
    - There is one full course on RiscV CPU verification, but partial attendance is also possible for those already familiar with CPU verification (but not familiar with RISC-V (see column 2 "Shortened version of the course")
3. RISC-V SoC verification (details) which focuses on the verification of an SoC which incorporates a RISC-V CPU. The main strategy deployed here is to use the RISC-V to run programs (written in C or assembler) to verify SoC integration and functionality. It uses an open-source test bench and students are expected to use that for the practical exercises.

Objectives

By the end of the course, participants should be able to:

1. Describe the current best-practice CPU and CPU-based SoC DV strategies.
2. Understand the main methodologies, tools and languages used in those best-practice CPU DV and CPU-based SoC strategies.
3. Apply those best-practice DV methodologies, tools and languages to a basic RISCV CPU design and a RISCV-based SoC design.
4. Analyse a "real" RISCV CPU design or RISCV-based SoC and propose a DV strategy.
5. Understand best-practice CPU and CPU-based SoC DV strategies so they can discuss DV topics confidently with colleagues.
6. Have sufficient understanding of RISCV CPU and a RISCV-based SoC DV tools and methodologies to contribute effectively to real projects.

Target audience

1. New recruits.
2. University students on placement.
3. Design engineers wanting to learn more about CPU and CPU-based SoC DV strategies.
4. Engineers wanting to transition their career to CPU and CPU-based SoC DV (or just want to learn more).
5. Managers wanting some understanding of CPU and CPU-based SoC DV.

Pre-requisites

Some experience of DV and System Verilog (SV) with the Universal Verification Methodology (UVM) would be beneficial.

Detailed content

Part 1: CPU and RISC-V Basics

1 Introduction to CPU (approx. 1 hour teaching)

1.1 Introduction to CPU architectures and Instruction Set Architectures (ISA)

- What is an ISA and why do we need them? (code/binary portability)

1.2 Introduction to CPU micro-architectures

- A simple CPU block diagram
- Basic operation of a simple CPU micro-architecture

2 How do we verify a CPU? (approx. 2 hours teaching)

2.1 Overall CPU verification strategies

- Contrast and compare different approaches to CPU verification, and their relative advantages and disadvantages
- Explain a hierarchical simulation-based approach which will be taught here

2.2 Unit level

- This can be done with standard SV-UVM test benches
- Give some basic examples from
    - fetch and branch prediction, decode and dispatch, integer execution, load store execution, floating point, instruction cache, data cache, MMU, CSRs. More specialised units may include vector unit, reorder buffer
    - Note detailed unit level verification is too specific to each design to be part of the course

2.3 CPU level

- Explain how this can be performed using binary machine code
- Explain how assembler code can be used to verify the CPU
- Explain how instruction stream generation can be used to generate the assembler
- Example of a UVM test bench for running the generated assembler (converted to binary) on the CPU

2.4 System level integration

- Brief introduction to the objectives of and techniques for verifying the integration of a CPU into an SoC (this is covered in more detail later in the course)

3 Basic CPU level verification and tooling (2 hours teaching + examples)

3.1 Instruction stream generators (ISGs)

- How instruction stream generation works
- The need for constrained pseudo-random generation
- The types of constraints needed

3.2 The “riscv-dv” (open source) tooling

- Overview of riscv-dv as an example of an ISG (this is covered in more detail later in the course)
- Some examples of running riscv-dv

3.3 Impact of adding your own instructions

- This will be explained but not covered in detail (as it is very design specific)

4 CPU microarchitectures for faster code execution (4 hours teaching)

4.1 Reviewing different microarchitecture styles

- For example, RISC vs CISC vs VLIW (this is covered in more detail later in the course)

5 Verifying CPU microarchitectures (4 hours teaching + examples)

5.1 Building functional coverage models for the microarchitecture

- Identify microarchitecture features to verify
- Define suitable functional coverage points
- Extend existing coverage model
- Run tests and review the coverage

5.2 Generate tests to hit the functional

- Review the current constraints
- Extend constraints to hit coverage points in the extended coverage model

6 riscv-dv practical exercises (estimated student independent study time = at least 8 hours, plus at least 1 hour classroom review)

- Overview of the exercises for extending coverage model and the constraints file, and then running tests to close coverage
- Explanation on how to work independently on the exercises
- How to get support and feedback

6.1 Joint classroom review and feedback on the exercises

- Joint session to review exercises

Part 2: RISC-V CPU verification

7 Introduction to RISCV (approx. 2 hours teaching)

7.1 RISC-V ISA Overview

- Understanding instruction formats
- RISC-V ISA modularity
- Privileges and the memory model

7.2 Assembly Language for RISC-V

- Example programs

8 Writing and running RISCV programs (approx. 1 hour teaching + examples)

8.1 Overview of RISCV software toolchains

- Flow diagrams for the software tool chains
- Flow for creating binary files from assembler and C

8.2 Introducing the RISC-V Assembler and Runtime Simulator (RARS)

- Writing and running examples assembler programs for the basic ISA
- Running them on RARS

8.3 Assembling C code into RISCV assembler

- Writing and running examples C programs for the basic ISA
- Running them on RARS
- Setting practical exercises for writing and running examples assembler and C programs on RARS
    - Consider including Metaware on RISC-V (Synopsys toolkit)

9 RISCV practical exercises (estimated student independent study time = at least 4 hours, plus at least 1 hour classroom review)

- Overview of the exercises for writing and running code for RISCV and running on RARS
- Explanation on how to work independently on the exercises
- How to get support and feedback

9.1 Joint classroom review and feedback on the exercises

- Joint session to review exercises

10 How do we verify integration of CPU in an SoC? (approx. 2 hours teaching)

10.1 System level integration verification

- Explain how software running on a mini-SoC can be used to verify the CPU
- Explain that different (higher performance) platforms might be needed to run the code
- Examples of compliance suites and benchmarks

11 Basic RISC-V level verification and tooling (2 hours teaching + examples)

11.1 The "riscv-dv" (open source) tooling

- Overview of riscv-dv
- Some examples of running riscv-dv
- Reviewing the code generated and run in simulation
- Reviewing the pass/fail
- Reviewing the coverage

11.2 How to generate and run your first riscv-dv test

- Run the tool, simulate the output, check the results, review the coverage for yourself

11.3 Reviewing the riscv-dv functional coverage model

- Review the functional coverage defined in riscv-dv

11.4 Reviewing the riscv-dv constraints model

- How constraints are defined in riscv-dv
- How to make changes and see the impacts of those changes

12 riscv-dv practical exercises (estimated student independent study time = at least 8 hours, plus at least 1 hour classroom review)

- Overview of the exercises for running the full riscv-dv flow
- Explanation on how to work independently on the exercises
- How to get support and feedback

12.1 Joint classroom review and feedback on the exercises

- Joint session to review exercises

13 Architectural compliance (2 hours teaching + examples)

13.1 Understanding architectural compliance

- What is architectural compliance?
- The different RISCV architectures and differences in compliance requirements
- Architectural compliance suites
- Detailed description of an example compliance suite

13.2 RISC-V compliance suites

- Running an example RISC-V compliance suite
- Reviewing the results

14 CPU microarchitectures for faster code execution (4 hours teaching)

14.1 Reviewing different microarchitecture styles

- For example, RISC vs CISC vs VLIW
- Why RISC and RISCV specifically?

14.2 The RISCV registers

- Overview of the register set
- Register Addressing Modes (e.g. direct, indirect, immediate)
- General purpose vs special registers

14.3 Pipelining

- What is pipelining and which problems does it solve?
- What problems does pipelining introduce?
- Techniques for overcoming these issues (e.g. out-of-order execution, branch prediction, register renaming)

14.4 Additional microarchitectures

- Superscalar processors
    - What is a superscalar processor and how are they designed?
- Multithreaded processors
- Vector processors and Vector/SIMD instruction set extensions
- Multi-core processors

15 Final session

- Review the main concepts
- Review the practical exercises and how they apply to real projects

16 Additional Materials

- Add material on how “Formal Verification” could be used

Part 3: RISC-V SoC verification

Full course syllabus

17 Introduction to SoC Verification (approx. 2 hours teaching)

17.1 Introduction to SoC verification

- The main SoC verification tasks and challenges
- The main differences between IP and SoC verification
- The main metrics and signoff criteria used in SoC verification

18 Introduction to the training SoC (approx. 2 hours teaching)

18.1 Introduction to the SoC being used in the course

- An overview of the architecture of SoC being used

18.2 The SoC verification environment

- Overview of the test bench
- The tools and flow for running tests and simulations
- How to add checkers using assertions and write self-checking tests
- Defining and implementing SoC functional coverage models

19 SoC practical exercises (estimated student independent study time = at least 4 hours, plus at least 1 hour classroom review)

- Overview of the environment for running RISCV code on the SoC
- Overview of the process of updating a test or write a new test and then run it on the SoC
- Exercises to run existing tests
- Exercises to update and run existing tests
- Exercises to write and run new tests
- Ensuring tests are self-checking
- Adding functional coverage

19.1 Joint classroom review and feedback on the exercises

- Joint session to review exercises

20 SoC practical debug exercises (estimated student independent study time = at least 2 hours, including classroom review)

- Finding bugs, triage and debug flow
- Exercise to run a test on an SoC with a known bug
- Exercise to triage and debug the known bug

21 How do we verify an SoC CPU? (approx. 3 hours teaching)

21.1 Overall SoC verification strategies

- Contrast and compare different approaches to SoC verification, and their relative advantages and disadvantages
- Explain the approach which will be taught here

21.2 SoC verification for specific SoC features

- SoC integration verification and tests
- SoC reset & boot (power on sequence) sequence verification and tests
- SoC clock control and clock gating verification and tests
- SoC power control verification and tests
- SoC interrupt verification and tests
- SoC system control verification and tests
- SoC use case verification and tests

22 SoC verification and signoff practical exercises (estimated student independent study time = at least 4 hours, plus at least 1 hour classroom review)

- Writing and running tests for a range of SoC features
- Collecting coverage and sign-off metrics
- Writing a SoC sign-off report

22.1 Joint classroom review and feedback on the exercises

- Joint session to review exercises

23 Final session

- Review the main concepts
- Review the practical exercises and how they apply to real projects

24 Additional Materials

- Additional material on how “Formal Verification” could be used