COURSE OUTLINE

Design Verification (DV) training

This course introduces participants to best-practice DV strategies so they can contribute effectively to real projects (see the objectives). It is aimed at new recruits, interns, and managers/engineers wanting to transition to DV, 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 training is spread over four weeks of live delivery preceded by a self-paced pre-course SystemVerilog module. Weeks 1 to 4 each have 2 hours per day of online lectures and practicals (see Table 1, more details can be found by following the links); the pre-course module is delivered as pre-recorded video and Moodle quizzes and is completed before Week 1. Exact times are agreed with the students in advance of the course.

A range of course materials are provided in advance to help participants to prepare (details). Different experience pathways though the examples and exercises are defined based on student prior experience and expertise. There are quizzes on Moodle (with automated grading) to provide feedback on understanding.

It is recommended that during the four weeks the students are given time for independent study on Moodle quizzes and exercises outside of the online session.

Table 1: Proposed content and timetable for the DV training with 2 hours of online training per day (four-week SV/UVM version)

Day	Week 1	Week 2	Week 3	Week 4
Mon	Lecture: "Lecture 1: Introduction", "Lecture 2: Simulation-based Verification" Practical: "Introduction to the exercises"	Lecture: "Lecture 8: SystemVerilog Assertions (SVA) in detail" Practical: "Practical 5: SVA"	Lecture: "Lecture 14: UVM stimulus path – virtual interfaces, sequence items, sequences, sequencers, test selection" Practical: "Practical 11: UVM – multi-port memory controller"	Lecture: "Lecture 19: UVM Register Abstraction Layer (part 1)" Practical: "Practical 15: Register model generation and integration"
Tues	Lecture: "Lecture 3: SystemVerilog language essentials and gotchas" Practical: "Introduction to Verilog/SV and simulation", "Running directed test benches"	Lecture: "Lecture 9: Coverage" Lecture: "Lecture 10: Functional coverage in SystemVerilog" Practical: "Practical 6: Coverage"	Lecture: "Lecture 15: UVM analysis path – TLM, scoreboarding, environment examples, UVM debug features, simulator overview" Practical: "Practical 12: UVM"	Lecture: "Lecture 20: UVM RAL (part 2) – access methods, prediction, adapters, mirroring, built-in sequences" Practical: "Practical 16: Register sequences and tests"
Wed	Lecture: "Lecture 4: SystemVerilog for verification – classes, dynamic data, interfaces, clocking blocks" Practical: "Practical 2: SystemVerilog classes and dynamic data"	Lecture: "Lecture 11: The FIFO example" Practical: "Practical 7: FIFO IP verification"	Lecture: "Lecture 16: Further UVM – reporting, command-line control, advanced factory, advanced TLM, debug interfaces" Practical: "Practical 13: Advanced UVM"	Lecture: "Lecture 21: Introduction to debug and Making debug efficient and effective", "Lecture 22: Additional topics" Practical: "Practical 17: Debug and qualification for IP verification"
Thurs	Lecture: "Lecture 5: Stimulus generation" Lecture: "Lecture 6: Randomisation and constraints in SystemVerilog" Practical: "Practical 3: Class-based random stimulus"	Lecture: "Lecture 12: Introduction to UVM" Practical: "Practical 8: UVM – ALU demonstration"	Lecture: "Lecture 17: Advanced UVM sequences and virtual sequencers" Practical: "Practical 14: SPI multi-agent demonstration"	Lecture: "Lecture 23: IP, subsystem and SoC" Practical: "FIFO subsystem verification", Practical: "CPU-based SoC verification"
Fri	Lecture: "Lecture 7: Checking and Assertion-Based Verification" Practical: "Practical 4: Verification checkers and assertions"	Lecture: "Lecture 13: UVM building blocks – constructors, factory, phases, configuration" Practical: "Practical 9: Building a basic UVM agent"	Lecture: "Lecture 18: UVCs and multi-agent environments" Practical: "Practical 10: UVC integration"	Lecture: "Lecture 24: Feature Extraction", Practical: "Feature Extraction Exercises" Lecture: "Lecture 25: The Verification Cycle and Verification in Practice", "Lecture 26: Summary"

Pre-course (self-paced, before Week 1): SystemVerilog language foundations delivered as pre-recorded video and Moodle quizzes – standard data types and literals, operators, procedural statements and procedural blocks, user-defined data types and structures, static arrays, hierarchy and connectivity, tasks and functions, and the main Verilog/SV "gotchas". Tool set-up and download of the exercise database are also completed in this period.

Contents

1 Course Objectives

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

1. Describe the best-practice DV strategies applied currently to semiconductor digital designs.
2. Understand the main methodologies, tools and languages used in those best-practice DV strategies.
3. Apply those DV methodologies, tools and languages to basic digital designs at IP, subsystem and SoC level.
4. Analyse a “real” semiconductor digital design (IP, subsystem and SoC) and propose a DV strategy.
5. Understand best-practice DV so they can discuss DV topics confidently with colleagues.
6. Have sufficient understanding of DV tools and methodologies to contribute effectively to real projects.

7. Write and debug SystemVerilog test bench code using classes, dynamic data structures, interfaces, clocking blocks, randomisation and functional coverage.
8. Build a complete UVM test bench from scratch, including agents, sequences, sequencers, scoreboards, the factory, phases and the configuration database.
9. Apply advanced UVM techniques including virtual sequences, multi-agent environments, UVCs and the UVM Register Abstraction Layer.

2 Target audience

1. New recruits.
2. University students on placement.
3. Design engineers wanting to learn more about verification and/or how to write re-usable verification.
4. Engineers wanting to transition their career to DV (or just want to learn more about DV).
5. Managers wanting some understanding of DV.
6. Verification engineers with directed-test experience who need to move to constrained-random SystemVerilog and UVM.

3 Pre-requisites

There are no pre-requisites, although some experience of programming (preferably with an Object-Oriented language) would be useful. The course is suitable for a wide range of people working in semiconductor design and development.

Participants are, however, expected to complete the self-paced pre-course SystemVerilog language module (data types and literals, operators, procedural statements and blocks, user-defined types and structures, static arrays, hierarchy and connectivity, tasks and functions, and the common Verilog/SV gotchas) before Week 1. The material and its Moodle quizzes are supplied in advance and no prior SystemVerilog knowledge is assumed.

4 Preparation

To allow participants to prepare for the course, the following will be shared in advance:

1. An overview of the practical exercises and explanations on how to run them
2. The database of practical exercises
3. PDF of the initial lecture slides
4. Preparation videos
5. A login in to Moodle to prepare
6. The pre-course SystemVerilog language module: pre-recorded lectures, worked examples and automatically marked Moodle quizzes
7. A tool and licence set-up checklist so that every participant can compile and simulate before Week 1

5 Course overview and structure

1. There are 40 one-hour live sessions delivered over four weeks, plus the self-paced pre-course SystemVerilog module, split as follows:
   - Online one-hour lectures covering the major DV topics (details).
   - Practical sessions (details).
   - Sessions on debug (details).
   - Dedicated lecture and practical time on the SystemVerilog language, UVM and advanced UVM including register modelling (details).
2. A large database of design and verification examples and exercises graded as 3 distinct levels of difficulty and complexity ready for running on a wide variety of simulators (details).
   - These can be downloaded so participants can run them on their machines using their licenses.
3. A number of pre-recorded videos (details) covering running the exercises and using various EDA tools.
4. Use of a FIFO example that builds through the course to show a full end-to-end example of verification.
5. Additional worked demonstrations used to teach UVM: an ALU, a multi-port memory controller and an SPI multi-agent environment.
6. Use of Moodle (details) for testing understanding of the course content and tracking progress on the course.

Note that participants will get additional support outside the lectures and exercises as needed or requested. Preparatory and follow-on reading will be suggested but neither are obligatory. There is both formative and summative assessment (details).

6 Course content details

6.1 DV lectures: Details

The online DV lectures will cover the following. These sessions will include online questions and polls to increase the level of interaction.

6.1.1 Introduction	• Course introduction. • What is Design Verification? • Verification complexities. • Preparing for Practical sessions on Hardware design and DV languages. • Verification tools. • The cost of bugs. • Overview of the simulators used on the course and their differences.
6.1.2 Simulation-based Verification	• Driving stimulus and Checking responses. • Observability and Controllability. • Black box vs. white box. • Compare/contrast with formal verification. • Verification hierarchy. • Introduction to the FIFO exercise.
6.1.3 SystemVerilog Language Essentials	• Standard data types and literals; 2-state versus 4-state types. • Procedural statements and procedural blocks. • Operators. • User-defined data types and structures (typedef, enum, struct, union, packed versus unpacked). • Static arrays. • Hierarchy and connectivity. • Tasks and functions – main differences and main use cases. • The main Verilog/SV “gotchas” that can cause bugs, including blocking versus non-blocking assignment and simulation region scheduling. • Note: this material is issued as the pre-course module and reviewed and consolidated in the live session.
6.1.4 SystemVerilog for Verification	• Overview of the main SV features introduced to support verification. • Queues, dynamic arrays and associative arrays. • Basic classes, the new constructor, handles and object lifetime. • Polymorphism and virtualisation – virtual methods, abstract classes and downcasting. • Interfaces, modports and how to use interfaces to improve verification productivity. • Virtual interfaces. • Clocking blocks, sampling and driving skew, and race avoidance. • Direct Programming Interface (DPI) – calling C from SystemVerilog and vice versa.
6.1.5 Stimulus generation	• Foundations. • Manual vs. Automation. • Independent vs coordinated generators? • Level of abstraction. • Online vs. offline generation. • Test length. • Randomness. • Stimulus for the FIFO. • Random stimulus in SystemVerilog – rand and randc, constraint blocks, randomize() and pre/post randomize.

	• Constraint solving order, implication, distributions and randomisation control. • Class-based random stimulus and randomisation of transaction objects.
6.1.6 Checking and Assertion-Based-Verification	• What and when to check. • Checking mechanisms including behavioural models. • Introduction to assertions. • Benefits and drawbacks of assertions. • Types of assertions. • Mutation testing. • Assertion libraries – to be added – add OVL as an example. • Checkers and assertions for the FIFO. • SystemVerilog Assertions (SVA) in detail – immediate versus concurrent assertions. • Sequences, properties, implication operators and repetition. • Sampling, the clocking of assertions and local variables. • Binding assertions to a design, assertion coverage and cover property.
6.1.7 Coverage	• Introduction. • Code coverage. • Functional coverage. • Coverage analysis and closure. • Coverage for the FIFO. • Coverage in SystemVerilog – covergroups, coverpoints, bins and cross coverage. • Sampling strategies, coverage options and weighting. • Merging coverage databases and reporting.
6.1.8 The FIFO example	• Applying the full verification cycle to the FIFO. • Signing off the FIFO at IP level.
6.1.9 Introduction to UVM	• General structure of a constrained random UVM test bench. • How to create suitable randomisation, sequences, checkers, coverage, assertions etc. • The key UVM structures for building such test benches • Introduction to the UVM library and the UVM methodology. • Overview of the structure of a UVM test bench. • The UVM class hierarchy – components versus objects. • Packaging and directory structure for a UVM environment. • Introduction to UVM classes and the new constructor.
6.1.10 UVM Building Blocks: Factory, Phases and Configuration	• Introduction to the UVM factory – registration, creation and type and instance overrides. • UVM phases – build, connect, run and the objection mechanism. • Basics of configuration and the configuration database. • Virtual interfaces, the top-level test bench and connecting to a DUT. • Test selection and building a test library.
6.1.11 UVM Stimulus and Analysis Paths	• Sequence items – fields, field automation and printing. • Sequences and sequencers, and the driver/sequencer handshake. • Transaction level modelling (TLM) – ports, exports, analysis ports and FIFOs. • Scoreboarding and reference models. • Functional coverage collection inside a UVM environment. • UVM environment examples and the FIFO UVM test bench demonstration. • Basic UVM debugging features.
6.1.12 Introduction to OSVVM	• General structure of a constrained random OSVVM test bench. • How to create suitable randomisation, sequences, checkers, coverage, assertions etc. • The key OSVVM structures for building such test benches
6.1.13 Further UVM	• Advanced UVM concepts using the FIFO test bench

	• The UVM reporting mechanism – verbosity, severity, catchers and log control. • Adding flexibility via the command line – parameters, plusargs and command line processing. • Advanced factory concepts. • Advanced TLM concepts. • Advanced interfaces used for debugging – assigning interfaces using the configuration database and binding interfaces.
6.1.14 Advanced UVM Sequences and Virtual Sequencers	• Alternatives to the uvm_do macros. • Multichannel sequences. • Building complex constraint models and sequence item generation. • Constraint layering and behaviour modification. • Building advanced sequences and virtual sequences. • Nested sequences and sequence properties. • Building advanced sequencers and virtual sequencers with dependencies between sequences. • Complex sequence selection.
6.1.15 UVCs and Multi-Agent Environments	• Multiple agents – active, passive and reactive. • Introduction to UVCs (Universal Verification Components). • Integrating multiple UVCs. • UVCs with multiple agents.
6.1.16 UVM Register Abstraction Layer (RAL)	• Introduction to register modelling and generating the register model. • User-defined front door, vendor extensions and back door paths. • Simple model integration. • Register access methods and memory access methods. • Register sequences. • Prediction – auto, explicit and passive. • Creating and modifying adapters. • Using the mirroring in scoreboards. • Custom register modelling and active monitoring. • Running built-in register sequences and tests. • IP-XACT 2014 and reg_verifier.
6.1.17 Additional topics	• Using lint to check design and test bench quality • Test bench qualification using mutation testing • Where/how can AI/ML possibly be applied in DV? • Some example applications • How to maximise re-use across projects and test benches. • How to improve debug productivity. • Improving productivity through the use of AI.
6.1.18 The Verification Cycle and Verification in Practice	• Verification planning. • Planning, milestones, tracking and completion criteria. • Add examples from Open HW project. • Verification metrics and closure. • Verification methodologies. • The verification cycle from start to finish. • Add Design for Verification. • Add Test bench qualification. • IP level verification in practise. • Higher level verification in practise
6.1.19 IP, subsystem and SoC (System-on-Chip)	• Levels of abstraction. • Differences in designs at IP, subsystem and SoC. • Bus interfaces and connecting IP. • Differences in verification at IP, subsystem and SoC. • Difference between ASIC and FPGA design flows. • Verifying the FIFO integration at subsystem and SoC levels.

6.1.20 Feature Extraction	• Some techniques to apply during feature extraction • Worked examples of feature extraction • Creating a Verification Plan from a feature extraction

6.2 Practical Sessions: Details

These practical sessions are aimed at giving first-hand experience on developing test benches using a wide range of languages and methodologies. Note that there are also opportunities to develop designs[1]. All practical sessions will include interactive question and answer sessions.

6.2.1 Introduction to the exercises	• Overview of the exercises. • How to run the exercises. • Some simple simulation exercises. • Overview of the videos. • Selecting which exercises to run and how to get feedback. • Overview of the supported simulators and how to compile, elaborate and run on each.
6.2.2 Introduction to Verilog/SV/VHDL and simulation	• Overview of Hardware Description Languages (HDLs). • Verilog, SV and VHDL (only the languages relevant to the audience will be covered) • SystemVerilog language exercises – data types, user-defined types and structures, static arrays, tasks and functions, hierarchy and connectivity. • Interface and modport exercises. • Finding and fixing the common Verilog/SV gotchas in supplied code.
6.2.3 SystemVerilog classes, dynamic data and randomisation	• Writing basic classes, constructors and class hierarchies. • Polymorphism and virtual methods in a test bench context. • Queues, dynamic arrays and associative arrays in a scoreboard. • Writing and debugging constraints; rand versus randc; constraint solving order. • Class-based random stimulus generation for the FIFO. • Clocking blocks and virtual interfaces in a class-based test bench.
6.2.4 Running directed test benches	• Running directed test benches in the exercises. • Adding some randomisation to those test benches.
6.2.5 Verification checkers and assertions	• Automated verification checks. • Scoreboards. • SV Assertions. • Writing concurrent assertions with implication and repetition operators. • Binding an assertion module to a design and collecting assertion coverage.
6.2.6 Coverage	• Functional coverage. • Code coverage. • Structural coverage. • Writing covergroups, coverpoints, bins and cross coverage in SystemVerilog. • Merging coverage databases and producing a closure report.
6.2.7 FIFO IP verification	• FIFO IP and test bench. • Can your test bench find the FIFO bugs?
6.2.8 UVM	• Developing a basic UVM test bench from scratch. • Guided demonstration: ALU UVM environment. • Guided demonstration: multi-port memory controller. • Building an agent – driver, monitor, sequencer and configuration object.

[1]Development of designs have been added to give participants the opportunity to better understand the designs they are trying to verify.

	• Using the factory, phases and the configuration database. • Connecting a virtual interface from the top-level test bench to the DUT. • Writing sequence items and sequences, and selecting tests. • Adding a scoreboard and an analysis path via TLM.
6.2.9 Advanced UVM	• Adding advanced UVM features into the FIFO test bench • Guided demonstration: SPI multi-agent environment. • Virtual sequences and virtual sequencers with dependencies between sequences. • Constraint layering and complex sequence selection. • Active, passive and reactive agents. • Integrating multiple UVCs into one environment. • Controlling a test from the command line; using the reporting mechanism and report catchers.
6.2.10 Register modelling exercises	• Generating and integrating a register model. • Front door and back door access; register and memory access methods. • Writing register sequences and running the built-in register test sequences. • Writing an adapter and configuring prediction.
6.2.11 OSVVM	• Developing a basic OSVVM test bench from scratch
6.2.12 Debug and qualification for IP verification	• FIFO test bench qualification. • FIFO debug exercises. • Using the UVM debug features – factory and configuration database dumps, topology printing and transaction recording.
6.2.13 FIFO subsystem verification	• Review of subsystem test bench solution.
6.2.14 CPU-based SoC verification	• Introduction to CPU-based test bench verification.
6.2.15 Feature Extraction Exercises	• Exercises on simple feature extraction

6.3 Debug content

The online DV lectures will cover debug. Specifically, the following content will be covered.

6.3.1 Introduction to debug and Making debug efficient and effective	• What is debug and where do bugs lie? • Debugging designs and verification environments. • Working with (not against) the design team. • Techniques to reduce debug time. • Ensuring a bug is fixed. • Making use of “bug clustering”. • Ensure my verification environment is designed for ease of debug? • How to improve debug productivity in SystemVerilog and UVM environments.

The practical sessions will also cover techniques and examples of how to perform debug in simulation. The exercise database contains examples and exercises where designs have bugs that need to be found using the EDA tools. There will be pre-recorded videos showing how to debug with the various EDA tools.

7 Moodle

Moodle (https://moodle.com/) is an online learning platform. Students have a personalised login for the course that allows them to

- Access course materials in advance
- Attempt online quizzes with automated marking and model answers to give them instant feedback on their understanding
- Update on their progress on the exercises
- Complete the pre-course SystemVerilog language module and its graded quizzes before Week 1

The lecturer is also able to use this to monitor participant understanding and progress.

8 Pre-recorded videos

There will be pre-recorded videos explaining how to run the exercises on all the supported EDA tools. They will also cover how to debug using the various debug examples in the database.

In addition, the full pre-course SystemVerilog language module is delivered as pre-recorded video, together with short videos walking through the ALU, multi-port memory controller and SPI multi-agent UVM demonstrations.

9 Database of examples and exercises

Participants will be given access to numerous small DV examples and exercises to support what is being taught in both the DV and the debug lectures. The exercises will be available in both SystemVerilog (some using UVM), VHDL (some using OSVVM and UVVM) and Python (using cocotb) that can be run on a wide range of simulators[2]. Exercises are also graded into "simple, medium, hard" to allow students to find exercises suited to their level of expertise:

- Small design exercises are provided for those participants who want to learn about design.
- Small test bench exercises are provided to allow participants to implement the concepts learned in the lectures.
- There is documentation and videos explaining how to run the examples on a wide range of tools.
- Standalone SystemVerilog language exercises covering data types, structures, arrays, tasks and functions, interfaces, classes, polymorphism, randomisation, assertions and functional coverage.
- A graded UVM exercise set built around the ALU, multi-port memory controller, FIFO and SPI multi-agent examples.
- An advanced UVM exercise set covering virtual sequences, multiple agents, UVCs and register modelling.
- The exercises will be kept small and simple due to time limitations:
  - There are directed test exercises and some randomisation is added.
  - There are separate exercises for assertions, coverage and debug.
  - There are small complete test bench UVM/OSVVM/UVVM and cocotb exercises.
  - There are separate exercises for classes, randomisation and constraints, and for register modelling.
  - Participants will mostly be expected to complete a small part of a design or change to a design or test bench.
  - There will be full solutions available for every exercise.

9.1 Access to exercises and DV tools

Participants will be able to download the exercises for running on their own machines with their own tools and licenses.

9.2 Mapping a path through the examples and exercises

A number of pathways though the examples and exercises for the participants, based on their level of experience.

A dedicated pathway is provided for participants who already know SystemVerilog and only need the UVM and advanced UVM content, and a second pathway for participants new to the language who follow the full sequence.

10 Assessment

The following assessment activities are available to participants:

- Participants who attend the full course will receive a TechWorks Academy certificate of attendance.
- There will be a multiple-choice end of course test[3].
- Moodle quizzes at the end of the pre-course SystemVerilog module and after each UVM and advanced UVM lecture.
- A final practical checkpoint in which participants extend a supplied UVM environment with a sequence, a scoreboard check and a coverage point.

[2]Including commercial simulators Cadence Xcelium, Siemens Questa, Synopsys VCS, and Aldec ActiveHDL. And free tools such as GHDL + GTKWave for VHDL and Verilator for SV UVM

[3]This is seen as higher value than an attendance certificate so attendees can still acquire a high value certificate even if they do not attend all sessions