

Advanced VHDL Verification – Made simple

Efficiency and quality is all a question of overview, readability, extensibility, maintainability and reuse, - and a good architecture is the answer.

*This applies for both Design **and** Verification*



Description

On average half the development time for an FPGA is spent on verification. It is possible to significantly reduce this time, and major reductions can be accomplished with minor adjustments. This is an intensive 3-day course on how to reduce development time and at the same time improve the quality.

The main differentiators between this and other similar courses are the focus on simplicity and the very structured approach to reuse - also inside a single project. We have seen and heard of many complex testbenches by various designers. A major problem with most of these testbenches seems to be that it gets too complex for everybody apart from the VHDL expert who designed it, – sometimes a person with a far more than average interest in the language or system details.

This course is based on the principles of ‘maximum cohesion & minimum coupling’ and ‘Divide and Conquer’, where the test case writer doesn’t have to know anything about the testbench implementation details, and the testbench implementer has a structured architecture all the way down. This approach to VHDL testbenches typically leads to man-hour savings of 20-60% and more, and is unique for this course.



ESA (the European Space Agency) is sponsoring the development of UVVM in order to improve efficiency and quality of FPGA VHDL verification.

Target participants

The course is aimed at FPGA designers and verification engineers with a good knowledge of VHDL and some experience with VHDL testbenches and verification. You should also have working knowledge with Questa, ModelSim, Riviera-PRO, Active-HDL or other simulator.



Doulos recommends UVVM for VHDL testbench architecture

Main Benefits

The main goal of this course is to show how you can achieve a far better testbench overview, readability, extensibility, maintainability and reuse – resulting in better quality and faster development.

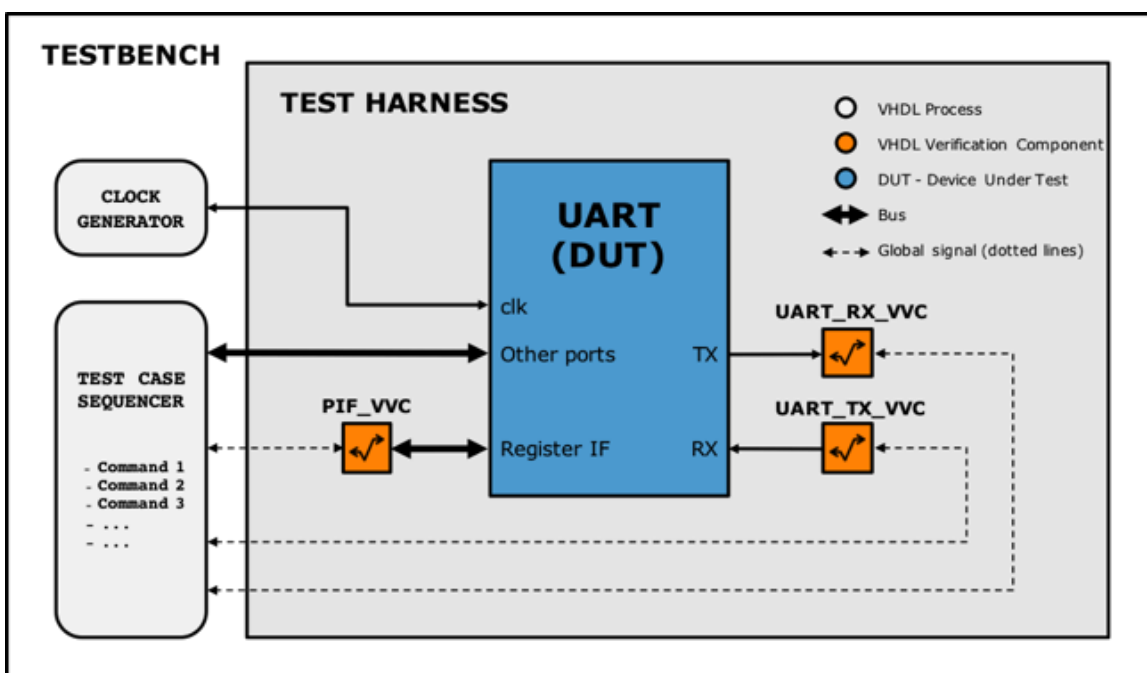
- **Overview:** So that anyone can easily understand the overall structure – just like a simple block diagram
- **Readability:** So that anyone can read the test driver and from that understand exactly what is going on
- **Extensibility:** Allowing fast and simple testbench changes to check new design modules and functionality
- **Maintainability:** Allowing fast and simple testbench changes to adapt to design changes
- **Reuse:** Between modules in a project; From modules to top-level in a project; From one project to another

What you will learn

It is actually possible for almost all companies to speed up the FPGA verification and at the same time improve the FPGA quality and fault coverage. Learn how to build your testbenches in a structured way, which is the key to overview, readability, extensibility, maintainability and reuse. Theory is mixed with practical examples and hands-on tutorials. The course will also cover important general verification issues like:

- Using sub-programs and various important VHDL constructs for verification
- Handling simple verification in a simple manner
- Making self-checking testbenches
- Using logging and alert handling
- Applying standard checkers for value and stability, and for waiting with a timeout for events
- Using simple procedure based transactions like `uart_transmit()` and `avalon_read()` for simple verification scenarios
- Making your own Bus Functional Model (BFM) – and adding features to speed up verification and debugging
- Getting a kick start on BFM's with UVVM's open source BFM's for Avalon, AXI4-lite/stream, UART, SPI, I2C, etc.
- Making directed or constrained random tests – and knowing where to use what - or a mix
- Learning to use the most important features from OSVVM for randomization and functional coverage
- Applying coverage driven tests in a controlled manner
- Using verification components and advanced transactions (TLM) for complex scenarios
- Target data and cycle related corner cases and verifying them
- Learning to use UVVM to speed up testbench writing and the verification process
- Getting a kick start on your testbench by using available UVVM Verification components for AXI4-lite, AXI4-Stream, Avalon MM, SBI, SPI, I2C, UART; - and use these as templates for your own VVC's
- Making a new UVVM Verification component in 30 minutes.
- Understanding and using Scoreboards
- Making an easily understandable and modifiable testbench even for really complex verification – and do this in a way that even SW and HW developers can understand them.

See below for a day to day agenda.



Guided Labs

The course will be approximately 50% theory and 50% hands-on. You will start by building a rather simple self-checking testbench, and then add procedures to simplify it, add Bus Functional Models to access your interfaces and add important functionality to these to speed up the verification process. Then you will write a test sequencer to control already available VHDL Verification Components (VVC) and experience how extremely easy that is. The next step is to add commands to an existing VVC and control constrained random stimuli and functional coverage, and finally you will generate and adapt your own VVC from scratch.

All participants must bring their own PC with their own preferred simulator (Questa, ModelSim, Riviera-PRO or Active-HDL). This allows an easy continuation after the course is finished.

UVVM with Constrained Random and Functional Coverage

UVVM (Universal VHDL Verification Methodology) and OSVVM (Open Source VHDL Verification Methodology) are both open source VHDL Verification Methodologies and Libraries. UVVM in combination with constrained random and functional coverage from OSVVM combines the best of two worlds into the 'Unified VHDL Verification Methodology' including:

- a) the most important UVM functionality, but extremely simplified for the user,
- b) the modular approach of a good FPGA design, with a hierarchical testbench structure that mirrors the design structure,
- c) a standardised VHDL testbench architecture and a standardised VHDL Verification Component architecture,
- d) a pure VHDL approach, where the user can just pick the functionality needed from well documented VHDL libraries, and
- e) the lowest possible user threshold for the functionality you need

UVVM will be used as example throughout the presentations and labs, but the principles taught and shown are general state of the art VHDL verification methodology. OSVVM will be used as example for constrained random and functional coverage. Examples will also show how OSVVM can be used **as is** inside and in parallel with UVVM. Hence you get basically 3 courses in one:

- General VHDL verification course with best practices for making good testbenches
- UVVM course – enabling the best possible VHDL testbench infrastructure and architecture
- OSVVM course – showing you how to use the powerful randomisation and functional coverage



Doulos recommends UVVM for VHDL testbench architecture

In a recent webinar Doulos presented several really good reasons for using UVVM as a basis for making structured and reusable VHDL testbenches. The quotes below are from John Aynsley, CTO in Doulos.

'the standardized testbench architecture that you find in UVVM is really useful for reuse' as you can in fact reuse a major part of your testbench from one project to another, and reuse the verification components exactly as is.

'within this framework you've got as much flexibility as you need' because you can define the test harness as you like, use any part of UVVM that you like (for instance just one or five specific VVCs) and mix and match with any legacy code or 3rd party verification IP.

'each VVC has a standardized structure' and this common architecture allows all VVCs to be easily understood, integrated, controlled and extended, - and of course makes it very simple to make new VVCs.

'It's relatively straight forward to write your own VVC', --- 'the first time it took me a little longer than 30 minutes to make my own VVC from scratch, .. but not much longer (given that the BFM's exist)', --- '(using the provided Python script...) this really is straight forward'

'Adding that monitor was a pretty straight forward thing to do' - thus showing that extending the VVCs with new functionality and new commands is quite easy.

'The UVVM structure is quite simple' - stating and showing that detailed knowledge of UVVM is not at all necessary to make new VVCs, make new structured testbenches or write good test sequences using the high level VVC commands. UVVM really makes it

Agenda

Day 1:

Making a Simple, Structured and Efficient Testbench, Step-by-step

- Testbench basics and basic testbenches, Building a simple TB from scratch, Simple TB architecture
- Testbench infrastructure, Logging, Alert handling, Checking values, Waiting for events with timeouts
- Procedures and functions, When to make subprograms, Using subprograms in TBs
- Simple randomisation, Transactions and Bus Functional Models

Advanced TB Infrastructure aspects

- Advanced logging and verbosity control, Advanced alert handling
- Pulse and clock generation, Advanced waiting-procedures
- Process synchronization between two or many processes

Making good BFM (Bus Functional Models, Transactions)

- Good BFM, TB and BFM synchronization, Making flexible subprograms,
- Normalisation and sanity check of subprogram inputs, Configuration of BFM behaviour
- Subprogram parameter simplifications

Introduction to more advanced TB architectures

- Regular and irregular data streams
- Parallel activity, Corner cases, Challenge of verifying a UART with corners

Labs: TB infrastructure and BFMs

- 1: Getting started, 2: Simple BFM, 3: Advanced BFM
- Procedures, logging, alerts, checkers, test harness, simple data communication using BFMs

Day 2:

Advanced TB architectures

- Limitations of typical TBs, Corner case categories, A structured TB architecture,
- Transaction level models, VVCs (VHDL Verification Components)
- Explaining a VVC, BFMs vs VVCs, VVC architecture, VVC reuse
- Structured TBs, Overview, Readability, Maintainability, Extensibility and Reuse
- VVC structure and extensibility, Handling multi-thread interfaces
- Making simple high level test case commands, Debugging, Hitting the corner cases
- Value and cycle related corner cases, Randomisation, Local sequencer
- Understanding and using Scoreboards

UVVM details (and general good verification methodology)

- Transaction commands – step-by-step, Command distribution
- Delta cycles, Non time consuming commands
- Commands queuing, VVC commands, Execution of VVC commands
- Controlling a VVC based TB, Procedure naming and parameters
- VVC configuration and status, Multicast and Broadcast
- Large data transfers, Handling split transaction interfaces (and out-of-order)
- BFM code walkthrough, VVC code walkthrough

Labs: UVVM VVC Framework

- 4: Getting started, 5: TB sequencer, 6: Making new commands
- Initiation, Logging in a VVC framework, Transaction commands
- Register access and UART examples, Simple synchronization, Simultaneous commands
- Adding VVCs to TB, Adding new commands to VVC, Controlling randomisation, Scoreboards

Day 3:

Randomisation and Functional Coverage – and Code coverage

- Code coverage, Functional coverage, Directed vs Random
- OSVVM randomization explained and exemplified
- OSVVM functional coverage explained and exemplified, Cover points,
- Using randomization and functional coverage inside UVVM

Various verification issues

- Documentation, Verification strategies, Verification approaches
- BFM and VVC vs SW for CPU access,
- Assertions, Checkers, Monitors
- Simple and advanced scoreboards

Reuse, Roadmap and Summary

- Testbench reuse at module and FPGA level, Reuse efficiency
- Debugging a VVC based TB, Roadmap for UVVM
- Efficient vs inefficient verification

Labs: UVVM VVC Framework

- 7: Making built-in checkers, Randomisation and Functional coverage, Making a VVC from scratch
- Make checker, Control checker
- OSVVM: Constrained randomisation, Randomisation inside a VVC, Intelligent random stimuli
Automatic coverage, Controlling randomisation and coverage
- Generate a new VVC template, Adapt VVC to your interface, Add commands

Presenter



Espen Tallaksen

Presenter and lab instructor at this course is Espen Tallaksen, the main architect behind UVVM. Espen is also the director and founder of Bitvis, the leading design centre for Embedded SW and FPGA in Scandinavia.

He has 30 years' experience from FPGA/ASIC development in various companies in Norway as well as NXP in Switzerland. He has given lots of courses on both Design and Verification, and technical presentation on various conferences around Europe. The fundamental message is always the same: Overview, Readability, Extensibility, Maintainability and Reuse are the key elements to Quality and Efficiency. Overly complicated design or verification systems should be avoided – even when they are structured. Simplicity – to the extent possible - should always be the target for any challenge.

More info:

- [Part 1: The testbench architecture](#)
- [Part 2: The testbench sequencer](#)
- [Part 3: The VHDL Verification Component \(VVC\)](#)

Registration:

See www.bitvis.no : Events

Please register via one of our course partners if you received the info/invitation from them.

