SynthWorks[SM]
Expert VHDL Training for Hardware Design and Verification

Advanced VHDL Testbenches and Verification

5 Days: 50% lecture, 50 % Lab **Advanced Level**

Course Overview

In Advanced VHDL Testbenches and Verification, you will learn the latest VHDL Verification techniques and methodologies for FPGAs, PLDs, and ASICs, including the Open Source VHDL Verification Methodology (OSVVM). You will gain the knowledge needed to improve your verification productivity and create a VHDL testbench environment that is competitive with other verification languages, such as SystemVerilog or 'e'. Our methodology works on VHDL simulators without additional licenses and is accessible to RTL engineers.

This class is structured to allow it to be taken either as individual modules or the full 5-day class. To ensure in-depth learning, 50% of the class time is devoted to hands on exercises and labs.

- In **Essential VHDL Testbenches and Verification (days 1 -3)** , you will learn to create structured transaction-based testbenches using either procedures or models (aka: verification IP or transaction level models). Both of these methods facilitate creation of simple, powerful, and readable tests. You will also learn about subprogram usage, libraries, file reading and writing, error reporting (Alerts and Affirmations), message handling (logs), abstractions for interface connectivity (records and resolution functions), model synchronization methods (barrier synchronization and others), verification data structures (scoreboards and FIFOs), directed, algorithmic, constrained random, and Intelligent Coverage random test generation, self-checking (result, timing, protocol checking and error injection), functional coverage, and test planning.
- In **Expert VHDL Testbenches and Verification (days 4-5)**, you will learn advanced topics including, modeling multi-threaded models (such as AXI4-Lite), advanced functional coverage, advanced randomization, creating data structures using protected types and access types, timing and execution, configurations and modeling RAM.

The lecture and labs in this class contain numerous examples that can be used as templates to accelerate your test and testbench development.

OSVVM, the world leading VHDL FPGA and PLD verification methodology, was developed by SynthWorks and grew up as part of this class. Each OSVVM package was used in class prior to its release into OSVVM, some of them for years. Taking this class from us helps further support the development of OSVVM and gives you the deep insight into the methodology that is only available from its developers.

Jumpstart your verification effort by reusing OSVVM packages for transaction based modeling, constrained and Intelligent Coverage™ random testing, functional coverage, error and message handling, interprocess synchronization, scoreboards, FIFOs, and memories.

VHDL Testbenches and Verification

Intended Audience

Suitable for digital (FPGA/ASIC/PLD) designers who are looking to improve their verification efficiency and effectiveness. Delegates should have a good working knowledge of digital circuits and prior exposure to VHDL through work or a previous course.

To take **Expert VHDL Testbenches and Verification**, delegates must have either completed **Essential VHDL Testbenches and Verification** or have instructor permission.

Learning Objectives

Essential VHDL Testbenches and Verification (days 1 -3):

- Create an OSVVM transaction-based testbench framework
- Write OSVVM transaction-based models (aka Verification IP or verification component)
- Simplify test writing using interface transactions (CpuRead, CpuWrite)
- Add error injection to interface transactions
- Implement a test plan that maximizes reuse from RTL to core to system-level tests
- Write directed, algorithmic, constrained random, and Intelligent Coverage random tests
- Write Functional Coverage to track your test requirements (test plan)
- Simplify error reporting using OSVVM's Alert and Affirm utilities
- Simplify conditional message printing (such as for debug) using OSVVM's log utilities
- Add self-checking to tests
- Use OSVVM's Generic Scoreboards and FIFOs
- Use OSVVM's Synchronization Utilities (WaitForBarrier, WaitForClock, ...)
- Understand VHDL's execution and timing
- Utilize OSVVM's growing library of Open Source Verification IP

Expert VHDL Testbenches and Verification (days 4-5)

- Write complex, multi-threaded verification components, such as AXI-Lite
- Use configurations to control which test runs
- Validate self-checking models
- Write AXI Stream Master and Slave Models
- Write models with interrupt handling capability
- Simplify memory model implementation using OSVVM's MemoryPkg
- Write protected types and access types
- Model analog values and periodic waveforms
- Advanced Coverage and Randomization techniques

Training Approach

This hands-on, how-to course is taught by experienced verification engineers. We prefer and encourage student and instructor interaction. Questions are welcome. Bring problematic code.

To reinforce lecture materials, approximately 50% of the class is labs. Both lecture and lab materials contain numerous examples that can be used as templates to accelerate your own test and testbench development.

Learn VHDL from a designer's perspective with SynthWorks.

Course Outline

Essential VHDL Testbenches and Verification (days 1 -3):

Day 1, Module TB1

- Overview
- Basic Testbenches
- Transactions and Subprograms
- Modeling for Verification
- VHDL and OSVVM IO
- Labs
  - Testing UartTx using Subprograms

Day 2, Module TB2

- Lab Review: Testing w/ subprograms
- Transaction-Based Models
- Elements of a Transaction-Based Model
- Generating and Checking Tests, Part 1
- OSVVM Library
- Labs:
  - UartTx using Models (Verification IP),
  - Adding Error Injection to UartTx,
  - Using OSVVM's Scoreboard

Day 3, Module TB3

- Lab Review: UartTx BFM
- Constrained Random Testing
- Functional Coverage
- Execution and Timing Issues
- Planning and Reuse
- Lab Review: Scoreboard, Random, Functional Coverage
- Labs:
  - Adding Functional Coverage
  - Adding Constrained and Intelligent Random
  - Reuse & Subblock Testing
  - UartRx using Models

Expert VHDL Testbenches and Verification (days 4-5):

Module XTB1

- Advanced Modeling w/ AXI lite Master
- Simulation Management and Configurations
- Data Structures and Protected Types
- Advanced Coverage Techniques
- Labs:
  - AXI Stream, Part 1

Module XTB2

- Advanced Randomization Techniques
- Generating and Checking Tests, Part 2
- Modeling RAM
- Interrupt Handling
- Validating Self-Checking models
- Labs:
  - AXI Stream, Part 2

Details

This class is a 5-day journey into VHDL coding styles, techniques, and methodologies that will help you become more productive at design verification and testbenches.

The heart of this approach is transaction-based testing. Transaction based testing abstractly represents an interface operation, such as a CPU read, CPU write, UART transmit or UART receive. In this class we learn two forms of transaction-based testbenches: a simple subprogram based approach, and a more capable component based approach (our preference). Both approaches use subprograms to initiate (call) a transaction. The subprograms are created in a package, allowing them to be reused by all testbenches.

In exploring the subprogram and component based approaches, we look in detail at a CPU (X86-lite) and a UART. For the X86-Lite, this includes protocol checking – such as receiving an

acknowledge when one is not expected. It also includes how to recover (timeout) when the DUT does not respond to an operation. For the UART this includes how to do error injection and error detection (that properly accounts for injected errors).

The OSVVM test harness (top level of the testbench) uses structural code, just like RTL code. The verification components (in the component based approach) use entities and architectures – again just like RTL. The verification components (also called models) can be written either RTL like or behaviorally. Generally this means that the both the testbench structure and verification components can be read (or written) by either verification or RTL engineers. While we believe where possible design and verification should be handled by separate engineers on a given project, we also believe that an engineer should be able to do either role – although this is common in the OSVVM / VHDL community, this is not common in the UVM/SystemVerilog community.

To simplify model connectivity, transaction interfaces are represented abstractly as a record with resolved elements (whose types are from the OSVVM ResolutionPkg). We call this an OSVVM interface and have been using it for 20+ years. IEEE 1076-2019 adds VHDL Interfaces to the language. VHDL interfaces are the future for these sort of connections. Until VHDL interfaces are implemented by simulators, OSVVM interfaces will serve as a prototype.

An OSVVM test case is an architecture of the test sequencer. The test case uses multiple processes - one for each independently running interface. A test consists of a sequence of calls to the transaction subprograms (again CPU read, ...). When necessary to synchronize actions done on independent interfaces, we use synchronization primitives from the OSVVM library.

A test suite is a collection of architectures of the test sequencer.

Writing tests by using transaction calls makes tests easier to write and read. More tests can be written in less time. With an explanation of what each transaction call does, anyone who can program can understand what the test does – this includes verification engineers, RTL design engineers, software engineers, or system engineers.

Additional efficiency is gained by encapsulating design specific sequences of transactions, such as "Initialize UART" and "Start DMA Operation" into a larger transaction. This helps with both test case generation and test case review.

In class we learn how to write directed, algorithmic, constrained random, and Intelligent Coverage random tests within the test sequencer. Transactions are the foundation for all of these test methods. This makes it easy to use the method that is best suited to achieve best test coverage – whether it be any specific one or a mix of them. The randomization (Constrained Random and Intelligent Coverage) methods covered in class use RandomPkg and CoveragePkg from the OSVVM library.

Since manually checking a test is tedious, error prone, and time consuming, making tests self-checking is important. In class we examine a number of different means of self-checking including result checking, protocol checking, timing checks, and error injection. The generic scoreboard introduced in this class is a FIFO like data structure used to facilitate result checking. The self-checking and scoreboard methods covered in class use AlertLogPkg and ScoreboardGenericPkg from the OSVVM library.

Coverage tells us when a test is done. There are several forms of coverage including structural (code coverage) and functional. Structural or code coverage is coverage detected and reported by

a simulator and is explored briefly in lecture and more detail in lab. Functional coverage goes beyond code coverage by looking for specific values or conditions to occur on an object or set of objects. The class covers functional coverage in detail and facilitates implementing it using CoveragePkg from the OSVVM.

Reuse saves projects time. Reuse requires good planning. We share methods for reuse of transactions, verification components, and test cases within a single project. It starts with a simple observation - while interface behavior (the signal wiggling) may change between subblock, core, and system-level tests, from a transaction call perspective, the test remains the same. We are still doing a CPU write. It still uses the same address and data information. Since writing test cases is a large investment, reuse of test cases reduces project schedules.

Customization

All of our courses can be customized to meet your specific needs. Contact us for details.

Contact

To schedule a class or for more information, contact Jim Lewis at +1-503-590-4787 or jim@SynthWorks.com.