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 are 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 coverage driven random test generation, self-checking (result, timing, protocol checking and error injection), functional coverage, representation of analog values and periodic waveforms (such as triangle or sine waves), 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, …),
- Model analog values and periodic waveforms,
- 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
- Understand VHDL's execution and timing
- 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
- Generating and Checking Tests, Part 2
- 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

- Execution and Timing Issues
- Advanced Randomization Techniques
- 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 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 bus functional model based approach. Both approaches use subprograms to initiate a transaction. A single test is a sequence of subprogram calls. A suite of tests is a collection of VHDL architectures. Using transaction calls makes tests easier to write and read, and more tests can be written in less time. The subprograms are created in a package, allowing them to be reused by all testbenches. Additional subprograms are used to encapsulate commonly used sequences of transactions (such as initialize UART and start DMA) further

improving effectiveness of writing and reading tests. While either the subprogram implementation or the transaction model may change between subblock, core, and system-level tests, the subprogram call interface largely remains unchanged allowing many tests and test models to be reused or pre-used at each of these levels - further improving your efficiency. To simplify using records as an interface channel, SynthWorks provides a resolution package as open source.

Another important topic is test generation. In class we learn how to create directed, algorithmic, constrained random, and Intelligent Coverage random tests. Using these methods, students learn how create tests that model a CPU, UART, FIFO, memory, arithmetic, and analog designs. Since transactions are the basis for all of these tests, it is easy mix any of the techniques together. This means that if a directed or algorithmic sequence can achieve test coverage faster, it can be used either by itself or mixed in with the randomization methods. The randomization and functional coverage packages that provide the foundation for constrained random and Intelligent Coverage random verification were released by SynthWorks into 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 scoreboard data structure used in the class, and provided as a reusable package, is a FIFO like data structure used to facilitate result checking.

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 the OSVVM coverage package (which was developed by SynthWorks).

Design verification requires a good test plan. We learn to write test plans that reuse interface waveforms, bus functional models, and test cases at the RTL, core, and system levels. Reuse removes redundancy in the verification effort and reduces the amount of time the project takes.

The SynthWorks testbench methodology uses the natural concurrency built into VHDL. Processes and models are used to create separate threads of execution for items that naturally run independently. When modeling statemachines in this approach, they can either be modeled concurrently, just like RTL code, or sequentially, just like software and OO approaches. To synchronize the separate processes, we use a synchronization primitive from a "library" of primitives contained in our open source utility package. The structure of the testbench is created with structural code just like RTL code. Compare this to other verification languages which rely on a complicated OO approach with "fork and join" to kludge in concurrency and methods for emulating elaboration and initialization that is already built into VHDL.

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.

Learn VHDL from a designer's perspective with SynthWorks.