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

Learn the latest VHDL verification methodologies for FPGA and ASIC design. Techniques include transaction level modeling (tlm), self-checking, scoreboards, memory modeling, functional coverage, directed, algorithmic, constrained random, and intelligent testbench test generation. Create a VHDL testbench environment that is competitive with other verification languages, such as SystemVerilog or 'e'. Our methodolgies work on VHDL simulators without additional licenses and are accessible to RTL engineers.

This course starts with simple testbenches and progressively increases the level of abstraction. Along the way students learn about subprogram usage, libraries, file reading and writing, modeling issues, transaction-based testbenches, bus functional models, transaction level models (tlm), record types, resolution functions, abstractions for interface connectivity, model synchronization methods, protected types, access types (pointers), data structures (linked-lists, scoreboards, memories), 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), timing and execution of code, test planning, and configurations. This class contains numerous examples that can be used as templates to accelerate your test and testbench development.

The final result is a system-level, transaction-based, self-checking test environment. Labs track with lecture giving students the opportunity to apply what they learn. The techniques you learn in this class give VHDL a similar capability to other popular verification languages.

What differentiates this class from other classes is that all of the techniques are implemented in VHDL and that core topics such as functional coverage, randomization, and data structures are covered in depth.

Jumpstart your verification effort by reusing our packages for transaction level modeling (TLM), constrained random testing, functional coverage, Intelligent Coverage™ testing scoreboards, and memories.

Intended Audience

VHDL Testbenches and Verification is recommended for experienced VHDL designers who are looking to improve their verification efficiency and effectiveness.

VHDL Testbenches and Verification

Course Objective

In this class, you will learn to:

- Be more productive at design verification and testbenches
- Create a transaction-based, system-level, self-checking test environment
- Implement interface functionality with either a subprogram or bus functional model
- Use subprograms to abstract interface actions (CpuRead, CpuWrite, UartSend)
- Write directed, algorithmic, constrained random, coverage driven random tests or a mixture of them.
- Write a test plan that maximizes reuse from RTL to core to system level tests.
- Reuse SynthWorks' packages for constrained random testing, functional coverage, memories, scoreboards, and interfaces.
- Model interface behavior with proper timing
- Model analog values and periodic waveforms
- Effectively use VHDL's file read and write capabilities
- Model RAM

Course Outline

Day 1, Module TB1

Testbench Overview
Basic Testbenches
Transactions and Subprograms
Modeling for Verification
VHDL IO

Day 2, Module TB2

Lab Review: Testing w/ subprograms
Transaction-Based Models (BFM)
Elements of a Transaction-Based BFM
Data Structures for Verification

Day 3, Module TB3

Lab Review: UartTx BFM
Creating Tests
Constrained Random Testing
Functional Coverage

Day 4, Module TB4

Execution and Timing
Configurations and Simulation Management
Advanced Coverage
Advanced Randomization

Day 5, Module TB5

Lab Review: Scoreboards, Randomization and Coverage
Test Plans
Modeling RAM
Transaction-Based BFM Part 2

Prerequisites

Students taking this course should have working knowledge of digital circuits and prior exposure to VHDL through experience or the course:

Comprehensive VHDL Introduction - 4 days

Learn VHDL from a designer's perspective with SynthWorks.

VHDL Testbenches and Verification

Other Recommended Courses

Students may also be interested in the following companion course:

VHDL Coding for Synthesis - 4 days

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 coverage driven constrained 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 package that provides the foundation for constrained random verification is provided as open source.

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. We cover functional coverage in detail and facilitate implementing it with our open source coverage package.

VHDL Testbenches and Verification

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. Either see our website or contact us for details.

Training Approach

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

Contact

To schedule a class or for more information, contact:

Jim Lewis
(800) 505-8435 / (800) 505-VHDL
+1-503-590-4787
jim@SynthWorks.com
http://www.SynthWorks.com

Learn VHDL from a designer's perspective with SynthWorks.