VHDL Expert Simulation & Synthesis

Course Description

This course expands the theoretical and practical know-how to write synthesizable and advanced test-benches through VHDL standard language.

The course goes into great depth and teaches the advanced features of the VHDL language to design complex projects, sophisticated test-bench, full generic design for reuse purposes, and advanced use of functions and procedures.

In addition, the course touches upon high frequency design problems, minimal area design, advanced timing analysis, advanced state machine design, design optimal arithmetic circuits, and much more.

The course emphasizes the most recent updates of the VHDL2008 standard, and guides how to employ them in new designs.

The course combines 50% theory with 50% practical work in every meeting. The practical labs cover all the theory and also include practical digital design.

This course also enriches digital engineers with many years of experience.

Course Duration

5 days

Goals

1. Write generic models at the highest level for synthesis
2. Use of pointers to generate linked-lists for advanced test-bench programs
3. Use generic functions and procedures for synthesis include operator and function overloading
4. Write VHDL programs with VHDL2008 standard
5. Write advanced test-benches that read and write from/to files
6. Use resource & functionality sharing techniques for most efficient code for synthesis
7. Design pipeline circuits with an emphasis on latency and throughput
8. Solve timing closure issues and optimize critical paths
9. Design optimal arithmetic circuits with an emphasis on algorithms
10. Design state machines for high frequency and reliability

Intended Users

Hardware engineers who develop FPGAs and would like to enhance their skills, in order to become experts with VHDL language

Previous Knowledge

- FPGA design
- VHDL

Course Material

1. Simulator: Modelsim
2. Synthesizer and Place & Route: Quartus Prime
3. Course book (including labs)

Table of Contents

Day #1

- **VHDL Coding Style for Performance & Small Area**
  - The synthesis process
  - Operator balancing & resource sharing
  - Functionality sharing
  - Quartus Prime synthesis options & attributes
  - High performance FSM & safe FSM
    - Mealy versus Moore design considerations
    - State encoding
    - Handling the unused states
    - Safe state machine
    - Encoding style settings
    - Moore output with look-ahead buffer
    - Using custom encoding styles

- **Pipelining**
  - Pipelining concept
  - Latency & throughput
  - Pipelining considerations
  - Pipeline balancing stages
  - Pipeline effectiveness calculations
  - Complex pipeline circuits design

- **Too Many Logic Levels Problems & Solutions**
  - Detect the problem
  - Analyze the problem
  - Solve the problem via VHDL or Synthesis/P&R tools

- **Fanout Problems & Solutions**
  - Detect the problem
  - Analyze the problem
  - Solve the problem via VHDL or Synthesis/P&R tools

- **Physical Synthesis**
  - Physical synthesis types

- Combinational logic optimizations
- Gate-level register retiming
- Asynchronous control signals optimizations

❖ **LAB #1: Apply Resource & Functionality Sharing techniques to achieve lower area utilization**
❖ **LAB #2: Fixing Advanced Timing Failures**
❖ **LAB #3: Explore different FSM encoding styles to achieve higher performance and safe behavior**

Day #2

- **Synthesis of Arithmetic Circuits**
  - Fast Counters Design
    - Ring counters
    - Johnson counters
    - LFSR counters
    - Pre-scaled counters
  - Fast Adders Design
    - Ripple-Carry adder limitation
    - Carry look-ahead adders
    - Carry select adders
    - Carry skip adders
    - Carry save adders
    - Wallace Tree implementation
    - 4:2 compressor adders
  - Fast Multipliers Design
    - Array multipliers
    - Radix-4 multipliers
    - Signed numbers multiplication
    - Booth algorithm multipliers

- **Area versus Frequency Tradeoff**
  - Functionality sharing for arithmetic circuits methodology
  - Sign magnitude adders
  - Adders-subtractors optimizations
  - Sign magnitude comparators
  - Signed addition with status

- Combinational adder-based multiplier

❖ **LAB #5: Apply different counting techniques to achieve higher performance**

❖ **LAB #6: Apply different adding & multiplying techniques to achieve higher performance**

Day #3

- **Design Reuse & Parameterized Design**
  - Requirements for reusable design
    - Design document
    - Verification strategy
    - Synchronous design techniques
    - Coding style
    - Test benches
    - Parameterizable
  - Design Reuse Components
  - Array attributes
  - Unconstrained Array
  - Clever use of an Array
  - Unconstrained Record
  - Converting loop into concurrent signal assignment
  - Subprograms in package
  - Generate statements for multiple architecture implementation
    - For generate
    - If generate
    - Case generate
    - Nested generate
    - Signals declaration in generate
    - Process declaration in generate
    - Controlling generate behavior from a package

- **Advanced Subprograms Implementation**
  - Review of functions and procedures
    - Rules
    - Variables behavior
    - Side effects

- Common and impure functions
- Synthesis implementation of subprograms
- Actual and formal parameters
- Resolution function
- Conversion function
- Function versus Procedure design consideration
- Design examples
- Subprogram overloading
- Operator overloading
- Using dot operator with same declaration in different packages

- **VHDL 2008 New Features Overview**
  - Why VHDL 2008?
  - Enhanced generics
    - Generic types
    - Generic constants
    - Generic in packages
  - Local packages
  - Nested packages
  - Package in subprogram
  - Generic lists in subprograms
  - Generic subprogram
  - Simplified sensitivity list
  - Simplified condition
  - "+"/"-" overload for scalar values
  - Unary reduction logic operators
  - The ?? condition operator
  - Matching relational operators
  - Minimum & maximum functions
  - Conditional & selected assignments
  - Matching case statements
  - Enhanced bit string literals
  - Context declarations
  - Unconstraint element types
  - New array types
  - Ports revision
  - Slices in aggregates
  - Select logic in port map

- Powerful generate statements
- Recursive generate statements
- Testbench enhancements with external names
- Force & Release
- Language support by tools

❖ **LAB #7: Write a generic function**
❖ **LAB #8: Apply parameterized design techniques**

Day #4

- **Introduction to Verification**
  - Effort spent on verification
  - Definitions
  - Verification vs testbench
  - Verification vs design
  - Re-convergence model
  - The human factor
  - Redundancy model
  - Equivalence checking
  - Model checking
  - Functional verification
  - Approaches comparison

- **Verification Tools**
  - Linting tools
  - Simulator tools
  - Verification IPs
  - Code coverage
  - Functional coverage
  - Assertions
  - Simulation-based verification flow
  - Formal verification flow

- **VHDL Timing Models**
  - Timing model importance
  - VHDL timing models overview

- Inertial delay
- Reject keyword
- Transport delay
- Inertial vs transport delay
- Delta delay

- **Elegant Testbenches**
  - Controllability $ observability
  - Stimulus & response
  - Verification limitation
  - Testbench types
  - Automated functional simulation
  - Assert statement
  - Self-verifying test bench
  - “Time-Slice” vectors
  - Breaking up stimulus blocks with procedures
  - Testbench with Record type
  - Testbench with data file
  - Using internal arrays for stimulus and results
  - Type conversions to STRING
  - Self-checking testbenches
  - Check absence of side-effects
  - Simulation with a golden design
  - Autonomous testbench

- **VHDL Advanced Features for Verification**
  - Modeling memories using static allocation
  - Modeling memories using dynamic allocation
  - Access type in VHDL
  - When to use access variables?
  - Example: memory array
  - Assignment of access variables
  - Access types for records & arrays
  - Access types for unconstrained arrays
  - Linked lists
  - Traversing the list
  - Memory management
  - Modeling memory with linked list
  - Shared variables

- Dual port RAM example
- Guidelines for using share variables
- Configuration & design hierarchy
- SDRAM example
- Using records in procedure calls
- PCI master example
- Generating random values
- Random number in VHDL
- What is OS-VVM?
- OS-VVM packages