VHDL Essentials Simulation & Synthesis

Course Description

This course provides all necessary theoretical and practical know-how to design programmable logic devices using VHDL standard language.

The course goes into great depth, and touches upon every aspect of the standard with directly connected to the topics needed in the industry today.

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.

The course begins with an overview of the current programmable logic devices and their capabilities, continues with an in-depth study of VHDL language with all of its structures, involves writing test-bench programs and employ a simulation tool. The course ends with a synthesis overview and emphasizes the difference between testing code and synthesizable code.

Course Duration

5 days

Goals

1. Become familiar with FPGA and CPLD families and their capabilities
2. Understand the design process from specification up to programming and final verification on board
3. Implement combinational and sequential processes
4. Build a hierarchy (bottom-up and top-down)
5. Write test-benches
6. Write generic code for design reuse
7. Understand coding style considerations for synthesis

Intended Users

Hardware engineers who would like start developing FPGA or CPLD

System engineers who would like to upgrade their professional skills.

Previous Knowledge

A basic background in digital logic

Course Material

1. Simulator: Modelsim
2. Synthesizer and Place & Route: Quartus Prime
3. Terasic Evaluation board DE0-CV
4. Course book (including labs)

Table of Contents

- **VHDL Basic Structures Overview**
  - Entity
  - Component
  - Architecture
  - Process
  - Functions & Procedures
  - Package & Package Body
  - Library
  - Configuration
  - Top down design

- **VHDL Design Units – Building a Hierarchy**
  - Building MUX from its primitives
  - Port Map
  - Test bench and simulation

Day #1 Labs

❖ Lab #1: Become Familiar with the Evaluation Board and VHDL

- Starting a new project in Quartus Prime
- Design entry using VHDL code (switches and led)
- Simulating the design circuit
- Compiling the designed circuit
- Pin assignment
- Programming and configuring the FPGA device

❖ Lab #2: Building Hierarchy

- Given a 3-bits binary adder design, simulate the design
- Design a 7-bit binary adder circuit using only 3-bits binary adder components
- Design a 7-segment decoder circuit to display the adding result on the board
- Simulate the design
- Compile the design
- Pin assignment
- Programming and configuring the FPGA device

Day #2

- **More on Entities**
  - Ports name
  - Direction
    - In
    - Out
    - InOut
    - Buffer
  - Data types
  - Generic
  - Generic map

- **Architecture Bodies**
  - Architecture Declarative Part
  - Behavioral Description
  - Data Flow Description
  - Structural Description

- **Concurrent Statements**
  - Simple signal assignment
  - Concurrent signal assignment
  - Implementation of signals
  - Conditional signal assignments
    - When-Else
    - With-Select
  - Examples of mux and encoder

- **VHDL Timing Model**
  - Inertial delay
  - Transport delay
  - Delta delay
  - Reject reserved word

- **Grammar and Declarations**
  - Data Types & Data Objects
  - Enumeration Types

- Attributes, Subtype
- Numeric Data Types (Integer & Real)
- Physical Data Types
- Composite Data Types
  - Array
  - Array attributes
  - Record
  - Aggregate
  - Multi-Dimensional Array

Day #2 Labs

❖ Lab #1: Dual Priority Encoder

- Design a dual-priority encoder that returns the codes of the highest or second highest priority requests
- Design the circuit in VHDL
- Write a testbench and simulate the design
- Inputs for the encoder will use the switches
- Display the two output codes on the 7-segments LED
- Compile the design, program the FPGA and verify its operation on board

❖ Lab #2: Arithmetic Logic Unit (ALU)

- Implement in VHDL 4-bit ALU with the following operations: add, sub, and, or, not a, not b
- The ALU should also produces zero and carry out flags
- Simulate the design
- Inputs for the ALU will use the switches
- Display the two output codes on the 7-segments LED
- Compile the design, program the FPGA and verify its operation on board

Day #3

- **Sequential Processing**
  - Process definition
  - Sensitivity list
  - Declaration area
  - Statement area
  - Sequential execution
  - Concurrent signal assignment versus process

- **Sequential Control Statements**
  - If-Elsif-Else statement
  - Case Statement
  - Loop statement
  - Next & Exit Loop control statements
  - Null
  - Assert statement
  - Wait statement

- **Package & Package Body**
  - Package declaration
  - Package Body declaration
  - Information hiding
  - Deferred constant
  - Reuse methodology
  - IEEE packages

Day #3 Labs

❖ Lab #1: Counter Design

- Develop an up-down 6-bit counter with a load and reset option that can count up-to 32
- When load = '1' then on the rising edge of the clock, 6 bit input data is loaded into the counter which keep counting from this new value
- If the maximum count is reached, then a 1-bit signal MAX is set high
- If the counter reaches zero, then a 1-bit output signal MIN is set high
- In either case the counter will stop until the direction of the counter is changed

❖ Lab #2: Using Loop Statement and a ROM

- Count the number of Zeros in the Odd indices of a 64 bit array
- Vector inputs will be read every clock from a 8x64 ROM
- Use Quartus IP catalog to instantiate a ROM in your VHDL program
- Read a new vector from the ROM on every rising edge clock
- Simulate the design
- Display the result on 7-segment led
- Compile the design, program the FPGA and verify its operation on board

Day #4

- **Process Behavior**
  - Signal assignment inside and outside a process
  - Signal and variable assignment differences
  - Process communication
  - Understanding the simulator algorithm
  - Passive process

- **Modeling Finite State Machines**
  - FSM concept
  - Mealy & Moore Models
  - HDL coding style
    - One process
    - Two processes
    - Three processes
    - Mealy & Moore
  - State encoding
    - Sequential
    - Johnson
    - One Hot
    - Two Hot
    - Defined by user
    - Defined by synthesis
  - Handling the unused states
  - Reset & Fail Safe Behavior
  - Interactive State Machines
    - Unidirectional
    - Bi-Directional

Day #4 Labs

❖ Lab #1: Fibonacci series

- Develop a concise VHDL algorithmic for Fibonacci series
- Device is activated with a START command (through button on the board)
- Compute the first 20 members of the series
- Simulate the design
- Display the last member on the 7-segment display
- Compile the design, program the FPGA and verify its operation on board

❖ Lab #2: Finite State Machine Design

- Design a state machine for a single input and single output Moore-type FSM that produces an output of 1 if the input sequence it detects is either “110” or “101” patterns
- Overlapping sequences should be detected
- Patterns will be implemented in ROM of 4x16 bits, where each rising edge clock, 1 bit will be read from ROM
- Repeat the same for a Mealy-type FSM
- Choose one-hot and binary for state machine encoding and analyze resource utilization results
- Simulate the design
- led should be used as indication that pattern is matched
- Compile the design, program the FPGA and verify its operation on board

❖ Lab #3: Finite State Machine Design

- Design a state machine with single input and single output
- The FSM should detect if a binary number can be divided by 3 without carry (for example, the pattern “011” or “100001” will produce z = 1)

Day #5

- **Subprogram: Functions & Procedures**
  - Subprograms introduction
  - Where to declare subprogram?
  - Functions definition & rules
  - Formal vs actual parameters
  - Impure functions
  - Resolution functions
  - Conversion functions
  - Standard functions
  - Procedures definition & rules
  - Matching object classes of formals & actuals in subprograms
  - Function vs procedure
  - Scalable width pipeline example

- **Constructing Testbenches**
  - Controllability & observability
  - Testbench benefits
  - Stimulus & response
  - Verification limitation
  - Interpretation of specification
  - Testbench types
  - Simulation types
  - Advantages and disadvantages of testbench types
  - Stimulus only testbench
  - Automated functional simulation
  - Test vector generation
  - Self-verifying testbench
  - Simplifying testbench with procedure
  - Time-slice vectors
  - Testbench with Record data type
  - Using internal arrays for stimulus & results
  - Self-checking testbenches
  - Basic I/O operations
  - File declarations
  - TEXTIO and std_logic_textio packages
  - Reading from files & standard input

- Writing to files & standard output
- Append mode
- File declared in subprograms
- Explicit open & close operations
- Handling failed operation

Day #5 Labs

❖ Lab #1: Sorting Algorithms

- Write a function that for a given array with 2n+1 integer elements, n elements appear twice in arbitrary places in the array and a single integer appears only once somewhere inside, finds the lonely integer
- For example: for the array input 3,5,4,4,3 the function should outputs 5
- Place the function in a package, then write an application with a call to your function in order to test it

❖ Lab #2: Verify State Machine Behavior

- Given the code for the state machine and testbench template, complete the testbench with declaration of 4x16 array in order to test 4 different test cases
- Write a process read each clock one bit and inject it to the FSM
  - After each array cell completion, you need to test the FSM from the beginning (each test vector is independent from the others)
- Given LFSR code, add it to your testbench and create a mechanism to inject each clock one bit to the FSM