# Verilog Essentials Simulation & Synthesis

## Course Description

This course provides all necessary theoretical and practical know-how to design programmable logic devices using Verilog 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 Verilog 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

### Day #1

- **Hierarchical Modeling Concept**
  - Top down and bottom up
  - Modules
  - Instances
  - Components of a simulation
  - Design example of a counter

- **Introduction to Verilog – Basic Concepts**
  - Lexical conventions (white space, comments, operators, number specification, unsized numbers, X or Z values, negative numbers, underscore characters and question marks, strings, identifiers and keywords, escaped identifiers)
  - Data types (value set, nets, registers, vectors, integer, real, time register, arrays, memories, parameters, strings)
  - System tasks and compiler directives (displaying information, monitoring information, stopping and finishing in simulation, compiler directives)

- **Modules and Ports**
  - Modules
  - Ports
  - Port connection rules (inputs, outputs, inouts, width matching, unconnected ports)
  - Connecting ports to external signals (connected by ordered list, connecting ports by name)
  - Hierarchical names

## Day #1 Labs

❖ **Lab #1: Become Familiar with the Evaluation Board and Verilog**
  - Starting a new project in Quartus Prime
  - Design entry using Verilog 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

- **Gate Level Modeling**
  - Gate types (and/or gates, buf/net gates, array of instances)
  - Gate delays (rise, fall, turn-off, min/typ/max values)

- **Dataflow Modeling**
  - Continuous assignments (implicit continuous assignment, implicit net declaration)
  - Delays (regular assignment delay, implicit continuous assignment delay, net declaration delay)
  - Expressions, operators and operands
  - Operator types (arithmetic, logical, relational, equality, bitwise, reduction, shift, concatenation, replication, conditional, operator precedence)

## 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 Verilog
- 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 Verilog 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

- **Behavioral Modeling**
  - Initial statement (combined variable declaration and initialization, combined port/data declaration and initialization, combined ANSI C style port declaration and initialization)
  - Always statement (always block)
  - Blocking assignments
  - Non-blocking assignments
  - Timing controls (delay based, regular delay control, intra-assignment, zero delay control, event based timing control, named event control, event OR control, level sensitive timing control)
  - Conditional statements (if-else, case, casex, casez, while loop, for loop, repeat loop, forever loop)
  - Sequential and parallel blocks (block types, parallel blocks, nested blocks, named blocks, disabling named blocks)
  - Generate blocks (generate loop, genvar, generate conditional, generate case)

## 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-bits 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 Verilog 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

- **Tasks and Functions**
  - Differences between tasks and functions
  - Tasks (task declaration and invocation, task examples, automatic tasks)
  - Functions (function declaration and invocation, function examples, recursive functions, constant functions, signed functions)

- **Modeling Finite State Machines**
  - FSM concept
  - Mealy & Moore Models
  - Verilog coding style
  - 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

- **Useful Modeling Techniques**
  - Procedural continuous assignments (assign and deassign, force and release)
  - Overriding parameters (defparam statement, module_instance parameter values)
  - Conditional compilation and execution
  - Time scales
  - Useful system tasks (file output, $fopen, $fdisplay, $fwrite, $fmonitor, $fstrobe, display hierarchy, strobing, random number generation, initializing memory from file, value change dump file)

## Day #4 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 output 5
- 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

# Day #5

- **Timing and Delays**
  - Distributed delay
  - Lumped delay
  - Pin-to-pin delays
  - Path delay modeling (specify blocks, parallel connection, full connection, edge sensitive paths, specparam statements, conditional path delays, rise, fall and turn-off delays, min, max and typical delays, handling x transitions)
  - Timing checks ($setup and $hold checks, $width check)

- **User Defined Primitives (UDP)**
  - UDP basics (parts of UDP definition, UDP rules)
  - Combinational UDPs (definition, state table entries, shorthand notation for don't cares, instantiating UDP primitives)
  - Sequential UDPs (definition, level sensitive, edge sensitive)
  - Guidelines for UDP design

- **Introduction to Synthesis**
  - What is Synthesis
  - Synthesis tools
  - Verilog programs for synthesis versus for simulation
  - Coding style and pitfalls
  - Demonstration

# Day #5 Labs

❖ **Lab #1: Final Exercise**

- Sort multiple buses via priority vector and output them according to the specification. You will be restricted by maximum latency allows and circuit size

- You have to synthesize your code and verify functionality through simulation
- The vectors will be stored in different RAM blocks and the solution will be demonstrated on the board