Verilog for Synthesis

Course Description

This course provides all necessary theoretical and practical know-how to write synthesizable HDL code through Verilog standard language.

The course goes into great depth and teaches efficient methods for writing Verilog code in a way that produces the precise digital circuit for various constraints like high frequency, low power, and minimal area.

The course covers the full synthesis process flow starting from reviewing methodologies, using development tools, adding constraints, implementing every Verilog structure in an optimal way, understanding the problems with bad coding style, learning the differences between simulation pre- and post-synthesis, analyzing critical paths, and reading and analyzing synthesis reports.

In addition, the course focuses on writing efficient code to save area, increasing frequency, designing for low power consumption, dealing with skew problems, working with external IPs, using attributes in Verilog code, implementing reliable, and high-speed finite state machines, solving design problems like high fanout and more.

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.

Course Duration

5 days

Goals

1. Understand the synthesis process flow and the difference between different tools
2. Learn precise coding style for combinational and sequential circuits
3. Understand synthesis of multi-files projects
4. Become familiar with the various constraints and adding them to project
5. Become familiar with design problems resulting from bad coding style
6. Design efficient circuits for minimal area or high frequency
7. Work with IPs and combining them in the synthesis flow
8. Produce reports, detect and correct timing problems

Intended Users

Hardware engineers who develop FPGAs and would like to enhance their skills, in order to understand synthesis limitations, to acquire better expertise on avoiding digital problems and to be to write efficient coding style for synthesis

Previous Knowledge

FPGA design, Verilog

Course Material

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

Table of Contents

Day #1

        - If-else synthesis
        - LUT equation analysis
        - 'X' and 'Z' values
        - LATCH problem
        - Don't care in synthesis
    - Inference from arithmetic and relational operators
        - Integer versus real
        - Arithmetic operators: + , -, *, /, %, **
        - Relational operators: >, <, =, /=, >=, <=
        - Shift operators: >>, <<, >>>, <<<
        - Constants in arithmetic and relational operators
    - Simulation versus synthesis behavior and synthesis guidelines

- **Operator Sharing**
    - Derivation of efficient HDL description
    - Reducing circuit size
    - Operator sharing using Verilog description
    - Operator sharing using synthesizer GUI options
    - Analyzing area and frequency
    - Tradeoff analysis
    - Complex operator sharing and synthesis tools limitations

Day #2

- **Procedural Statements Synthesis**
    - Inference from within procedural statement introduction
    - General guidelines
        - Using variables
        - Case versus IF-ELSE
        - Combinational circuits sensitivity list
        - Sequential circuits sensitivity list
    - Inference from simple assignment statements
    - Inference from IF-ELSE and IF-ELSE IF statements
        - Prioritization and dependency
        - Latch problem
        - Variable not initialized
        - Full versus partial sensitivity list
        - Combinational versus sequential If statements

        - Operator sharing within procedural blocks
        - Nested IF statements synthesis
        - Using don't cares for final else clause
        - Analyzing results on different synthesizers
    - Inference from Case statements
        - General guidelines for synthesis
        - Combinational versus sequential sensitivity list
        - Don't care in Case statements
        - Latch problem
        - "Full" case attribute
        - "Parallel" case attribute
    - Inference from loop statements
        - Serial loop versus parallel loop synthesis
        - Loop index
        - FOR loop versus while, forever, repeat loop for synthesis
    - Combinational circuit design examples
        - Gray code incrementor
        - Programmable priority encoder
        - Signed addition with status
        - ALU
        - CRC

Day #3

- **Nonblocking Assignment for Synthesis**
    - Verilog race conditions
        - IEEE Verilog standard determinism
        - General guidelines to avoid race conditions
    - Nonblocking assignments coding guidelines for synthesis
        - The Verilog stratified event queue
        - Self-triggering always blocks
        - Combinational and sequential logic in the same always block
        - Blocking and nonblocking assignments in the same always block problem
        - Assignments to the same variable from more than one always block
        - Assignments using #0 issues

- **Coding style Guidelines**
  - Sensitivity List
    - Incomplete sensitivity list, simulation pre and post synthesis
    - Complete sensitivity list with mis-ordered assignments
  - Closed feedback loop problem
    - Combinational loops
    - Combinational loops overcome
  - Synchronous Circuits Inference
    - latch versus flip-flop inference
    - Synchronous and asynchronous reset
    - Load and enable signals
    - Incorrect control signal priority
    - Shift registers
    - Counters
    - Single port memory
    - Dual port memory
    - Initializing memory contents
    - Inferring ROM
    - Inferring multiplier and DSP
    - Adder trees

- **Finite State Machine Synthesis**
  - State machine structure (Mealy and Moore block diagram)
  - State encoding (Auto, one-hot, binary, Johnson, two-hot, Gray, User specific)
  - State machine implementation in Verilog (one, two or three always blocks)
  - Bad coding styles for state machine
  - Specifying encoding style in Verilog and in synthesizer tool
  - Registered outputs with and without latency
  - Using custom encoding styles
  - State machine attributes
  - Analyzing encoding style area and performance
  - High reliability safe state machines using attributes, handling illegal states

Day #4

- **Timing Analysis of a Synchronous Sequential Circuits**
  - Introduction to timing constraints and synchronous design techniques
  - Synchronized versus unsynchronized I/O
  - Use I/O FF
  - Setup time violation and maximal clock rate
  - Synthesis static analysis formula
  - Hold time violation
  - Output related timing considerations
  - Input related timing considerations
  - Poor design practices and their remedies
    - Misuse of asynchronous signals
    - Misuse of gated clock
    - Misuse of derived clocks
    - Global clock
    - Clock multiplexing
    - Clock skew
  - Analyzing critical paths in details
  - Register changes during synthesis
  - Synthesis static timing analysis versus Place & Route static timing analysis
  - Fan-out and long combinational chain timing problems
    - Logic duplication
    - Automatic fan-out control
    - Shift register example
  - Using PLL in the system
  - Physical Synthesis versus logical synthesis

Day #5

- **Synthesis of Large Projects**
  - Reuse methodology
  - Parameterized code
    - Parameters review

        - Generate constructs (if-generate, case-generate, loop-generate, nested generate)
    - Synthesis of functions & tasks
    - Compiler directives `define, `include, `ifdef, `else, `endif

- **Integrating IP Core**
    - IP core generation and integration into Verilog code
    - IP black box constraints

- **Maximize Clock Rate**
    - Introduction to pipeline
    - Latency and throughput
    - Pipelined combinational circuits
    - Pipelining considerations
    - Pipeline balancing
    - Effectiveness of pipeline
    - Adding a pipeline in Verilog
    - Analyze clock rate in pipelined design
    - Complex pipeline circuits
    - Retiming

P.O.B. 803 Kfar Saba 441080 ISRAEL | P +972-52-5816791 | F +972-77-4702742 | ContactUs@HandsOnTraining.co.il | HandsOnTraining.co.il