Advanced Verilog for High Productivity

Course Description

This course provides all necessary theoretical and practical know-how to write synthesizable Verilog code high productivity in an efficient way.

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 starts by introducing power consumption challenges and how to write efficient Verilog code in order to decrease power in ASIC designs, including resource sharing, functionality sharing, state machine encoding, minimizing transitions on bus, clock gating, how to control counters, retiming and much more.

In addition, the course focuses on writing efficient code to save area, alternative algorithms for arithmetic circuits including counters, adders, multipliers and comparators.

For high frequency design, the training goes into pipeline technique including efficiency, balancing, advantages and disadvantages, skew and high fanout issues.

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

3 days

Goals

1. Understand how efficient coding can decrease power consumption
2. Design high-speed arithmetic circuits
3. Design efficient circuits for minimal area or high frequency
4. Introduce System Verilog features for synthesis

Intended Users

Hardware engineers who develop ASICs and would like to enhance their skills, in order to understand synthesis limitations, write efficient coding style for low power, low area or high frequency.

Previous Knowledge

- Verilog
- Digital Design

Course Material

1. Course book

Table of Contents

Day #1

- **Introduction to Low Power Design**
  - Power consumption review
  - Domains of low power design
  - Low power design at RTL and gate levels

- **Low Power Design Techniques**
  - Signal coding
    - One hot
    - Gray
    - Bus inversion
    - Hamming distance
    - Sign magnitude representation
    - FSM encoding
  - Clock gating
    - Power/energy optimization space
    - Reducing active power at system level
    - Clock gating
    - Circuit-level activity encoding
    - Eliminate glitches
    - Clock gating through FSM
  - Double-edge clocking
    - Energy consumption ratio
    - Low power optimization
  - Glitch reduction
    - Glitch propagation
    - Glitch reduction with FF
    - Glitch reduction with multi-phase clocking
    - Glitch reduction with delay balancing
    - Glitch reduction with SOP
  - Operand isolation
    - Blocking arithmetic data path
    - Decoder example
    - Control signal gating

- Pre-computation
- Concurrency insertion
  - Concurrency vs redundancy
  - Concurrency insertion techniques
- Parallelism & pipelining
- Algorithm level
  - Reordering inputs
  - Apply different algorithms

- **Resource & Functionality Sharing**
  - Derivation of efficient HDL description
  - Resource sharing definition
  - Operator sharing using Verilog description
  - Balancing operators
  - Multipliers balancing
  - Counter efficient design
  - Functionality sharing definition
  - Analyzing area and frequency of various design examples

- **Pipelining**
  - Pipelining concept
  - Latency & throughput
  - Pipelining considerations
  - Pipeline balancing stages
  - Pipeline effectiveness calculations
  - Complex pipeline circuits design
  - Timing and area analysis of a pipelined circuit
  - Retiming and physical synthesis techniques

Day #2

- **Synthesis of Arithmetic Circuits**
  - Adders Design
    - Basic adders
    - Carry chain adders
    - Carry skip adders
    - Carry select adders
    - Carry look-ahead adders
    - Carry save adders
    - Optimization of adders
  - Counters Design
    - Parallel counters
    - Up counters versus down counters
    - Ring counters
    - Johnson counters
    - Pre-scaled counters
    - LFSR counters
  - Subtractors and adder-subtractors
  - Multipliers Design
    - Basic multiplier
    - Radix-4 multipliers
    - Booth algorithm multipliers

- **Area versus Frequency Tradeoff**
  - Functionality sharing for arithmetic circuits methodology
  - Sign magnitude adders
  - Adders-subtractors optimizations
  - Sign magnitude comparators
  - Absolute circuits
  - Combinational functionality sharing

- **Advanced Finite State Machines**
  - Unidirectional and bi-directional complex state machines
  - Mealy versus Moore design considerations
  - State machine encoding
    - Advantages of each method and its Verilog implementation

- One-hot
- Two-hot
- Gray-code
- Johnson
- Sequential
- Random
- User defined

- Resource sharing of FSM
- Area speed and device resource utilization
- Optimizations of FSM
  - Outputs decoded in parallel output registers
  - Outputs decoded within state bits
  - One hot versus gray code encoding style performance
  - Using custom encoding styles
- High reliability FSM
  - Definition of fault tolerant design
  - Using attributes
  - Using constants
  - Handling illegal states
  - FSM SEU definition
  - Applying hamming algorithm for error detection and correction
  - Using “safe” attribute

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

Day #3

- **Introduction to SystemVerilog for Synthesis**
  - Verilog vs SystemVerilog: standards review
  - New SystemVerilog features
    - New data type: “logic”
    - New Always blocks
    - FSM enumerated type
    - $log2c() function
    - Two-dimensional port declaration

- **Synthesizing SystemVerilog**
  - Data types synthesis
    - Value sets
    - Net types
    - Variable types
    - Packed arrays
    - Unpacked arrays
    - Copying arrays
    - Passing arrays through module ports and to tasks/functions
    - Array query system functions for synthesis
    - Enumerated types
    - Structures
    - Unions
    - Type definitions
  - Parameterized models
  - Packages and $unit
  - Always_comb, always_ff, always_latch
  - Case equality operators (==?, !=?)
  - Set membership operator (inside)
  - Streaming operators
  - Increment/decrement and assignment operators
  - Casting
  - Case…inside
  - Unique, unique0, and priority decisions
  - Do…while new loop for synthesis
  - Multiple loop control variables in for loop
  - Break and continue

- Foreach loop
- Tasks and functions enhancements
  - Void functions
  - Formal arguments default to input
  - Arrays, structures, unions, and user defined types as formal arguments
  - Pass by name in calls
  - Using return keyword
  - Parameterized task/function arguments using static classes
- Relaxed Module ports rules
- Interfaces keyword
- Ending names
- `begin_keywords and `end_keywords
- Vector fill tokens
- Constant variable
- Timeunit and timeprecision
- Expression size functions ($clog2, $bits)
- Assertions