Intel® FPGA Technical Training

Designing for High Performance

Course Description

This course provides all necessary theoretical and practical know-how to increase FPGA performance through various advanced techniques.

The course goes into great depth and 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 combines 50% theory with 50% practical work in every meeting. The practical labs cover most of the theory and also include practical digital design.

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

Course Duration

2 days

Goals

1. Write an efficient code to maximize FPGA performance and synthesis tools utilization
2. Design pipeline circuits with an emphasis on latency and throughput
3. Solve timing closure issues and optimize critical paths
4. Design optimal arithmetic circuits with an emphasis on algorithms
5. 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 increase performance for logical, arithmetic, and state machine designs as well as timing problems that requires techniques such as pipeline, logic duplication, and alternative algorithms.

Previous Knowledge

FPGA design, VHDL, TimeQuest

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**
  - Operator balancing & resource sharing
  - Introduction to synthesis of arithmetic circuit
  - Duplicating registers to resolve Fanout problems
    - Detect the problem
    - Analyze the problem
    - Solve the problem via VHDL or Synthesis/P&R tools
  - Partitioning at register boundary methodology
  - Introduction to pipelining for high speed design
  - Case versus IF-ELSE in complex statements
  - Critical path optimization and reorder paths
  - Synchronous versus asynchronous reset design style

- **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

    - **LAB #1: Optimize VHDL code for high frequency**
    - **LAB #2: Use pipeline technique to increase complex circuit performance**

Day #2

- **Synthesis of Arithmetic Circuits**
  - FPGAs architecture arithmetic support
    - LUT modes
    - Built in adders
    - DSP blocks
  - Adders Design
    - Basic adders
    - Carry chain adders
    - Carry skip adders
    - Carry select adders
    - Carry look-ahead adders
    - Prefix adders
    - Multi-operand adders
    - Long operand adders
    - Carry save adders
    - Optimization of adders
  - Counters Design
    - Parallel counters
    - Up counters versus down counters
    - Ring counters
    - Johnson counters
    - LFSR counters
  - Subtractors and adder-subtractors
  - Sign magnitude adders and subtractors
  - Termination detection
  - Multipliers Design
    - Basic multiplier
    - Sequential multiplier
    - Ripple-carry multiplier
    - Carry-save multiplier
    - Multipliers based on multi-operand adders
    - 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 VHDL implementation
      - One-hot
      - Two-hot
      - Gray-code
      - Johnson
      - Sequential
      - Random
      - User defined
  - Resource sharing of FSM
    - 1/2/3 processes implementation
  - Area speed and device resource utilization
  - Timing analysis of FSM
  - Optimizations of FSM
    - Outputs decoded in parallel output registers
    - Outputs decoded within state bits
    - One hot versus gray code encoding style performance
  - 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
    - Example of dead lock problem

❖ **LAB #3: Optimize arithmetic circuit performance**
❖ **LAB #4: Increase State Machine performance**