HandsOn Training

Intel® FPGA Technical Training

Intel FPGA Designer Program

Course Description

This training program provides all necessary theoretical and practical know-how to design Intel FPGA using Verilog standard language and Quartus Prime software tools.

The course intention is to train computer and electronics engineers from scratch to a practical work level.

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

The course combines 50% theory with 50% practical work in every meeting with Terasic DE0-CV evaluation board. The practical labs cover all the theory and also include practical digital design. The program provides extra labs/mini projects for homework between meetings.

The first part of the program (5 days) 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 second part of the program (7 days) starts with an overview of Quartus Prime features, projects types and management, design methodology, and using IP cores from the IP catalog. Qsys system integration tool, state machine editor, memory editor, Altera SD for OpenCL, and DSP Builder are also introduced in high level. The course continuous with Quartus Prime compilation flow, incremental compilation concept, working with messages, viewing compilation reports, RTL and technology views, state machine viewer, and how to use the chip planner tool, I/O planning with the pin planner, with the BluePrint Platform Designer, and programming and configuration of FPGA. In addition attendee will learn how to write code for synthesis and employ recommended digital design practices.

The course also teaches the TimeQuest Tool, how insert timing constraints, analyze design timing issues and solve them, and how to debug the design on board use various debugging tools.

The third part of the program (3 days) covers advanced digital design concepts and optimizations. The training teaches how to design multiple-clock domains with various synchronization techniques, and measure the MTBF of the solution in TimeQuest. In addition this part teaches how to increase design frequency with pipeline techniques, physical synthesis, fast arithmetic algorithms, and methods to decrease design area with resource & functionality sharing techniques.

At the end of the program engineers will feel confidence to design simple & complex FPGA projects by their own.

Course Duration

15 days: Verilog (5 days), Quartus Prime (7 days), Optimization (3 days)

Goals

1. Become familiar with Intel FPGA/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. Understand coding style considerations for synthesis
7. Become familiar with TimeQuest and sdc files to insert timing constraints
8. Employ efficient design methodologies for high frequency or minimal area
9. Configure and embed IP in the design
10. Produce and analyze reports
11. Identify and fix timing issues
12. Utilize the FPGA architecture in the most efficient way with proper RTL coding style
13. Design pipeline circuits with an emphasis on latency and throughput
14. Close design timing and optimize critical path
15. Configure and use PLL in the design
16. Design a reliable multi-clock domains system with synchronization circuits
17. Design a reliable reset circuits
18. Optimize the design with synthesis and Place & Route tools
19. Program the FPGA on board

20. Use SignalTap II, SignalProbe, and other on-board verification tools

Intended Users

Hardware engineers who would like start developing projects based on Intel FPGAs

System and computer 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

Part I – Verilog Package (5 Days)

Day #1

- **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

Part II – Quartus Prime Package (7 Days)

Day #6

- **Quartus Prime: Project Management**
  - Project type
  - Altera Design Store
  - Project files and folders
  - Constraint files & assignment priority
  - Project and IP management
    - Project archive
    - Project copy
    - Revisions
    - IP components
    - Upgrading IP components

- **Quartus Prime: IP Cores**
  - Overview
  - IP base suite
  - IP Megacores
  - IP Catalog
  - IP settings
  - Parameterizing IP
  - MegaWizard Plug-In Manager and Parameter Editor

- **Quartus Prime: Compilation Process**
  - Processing options
  - Notification center
  - Compilation design flows
  - Incremental compilation concept
  - Rapid recompilation
  - Message window
  - Viewing compilation results
  - Netlist viewers
  - State machine viewer
  - Chip planner
  - Resource property editor

- **Quartus Prime Modeling Finite State Machines and Memories**
  - State machine editor
  - From SMF to HDL
  - Memory editor
  - Create memory initialization file
    Using memory file in design

- **Quartus Prime: Assignment Editor**
  - Synthesis and fitting control
  - Synthesis settings
  - Fitter settings
  - Assignment editor features
  - Design assistance

- **Quartus Prime: I/O Management**
  - Pin planner
  - Assigning pin locations using Pin Planner
  - Pin Legend and Pin Planner views
  - Pin migration view
  - Clock/PLL views and support
  - Pin Planner tasks & report windows
  - Reserved and unused I/O pins
  - Show fitter placements
  - Back annotation
  - Verifying I/O assignment
  - I/O planning with the BluePrint Platform Designer
  - Import/export via CSV
  - .qsf editing & scripting
  - Global pin settings

- **Quartus Prime: Programming the FPGA**
  - Programming files
  - Programming file conversion
  - Programmer GUI (hardware setup, JTAG settings, JTAG chain debugger tool)

Day #6 Labs

❖ Lab #1: Pipelined Multiplier Design

- Build an 8x8 multiplier using the IP catalog
- Create .hex file using the Memory Editor
- Create a 32x16 RAM from the IP catalog
- Connect all components in VHDL top level file
- Perform a full compilation
- Locate information in the compilation report
- Explore cross-probing capabilities by viewing logic in various windows (RTL view, technology view, and chip planner)

❖ Lab #2: Revisions & Pin Assignments

- Create new revision to store new constraint settings
- Make design constraints using the assignment editor
- Assign I/O pins and perform I/O assignment analysis
- Back annotate pin assignments to lock placements
- Use the pin migration view to see the effect of device migration on I/O assignments

Day #7

- **Introduction to Timing Analysis**
  - TimeQuest tool overview
  - Basic steps to using TimeQuest (generate timing netlist, enter SDC constraints, update timing netlist, generate timing reports)
  - Using TimeQuest in Quartus Prime flow
  - Timing analysis basics (Launch Vs Latch edges, setup and hold times, data and clock arrival time, data required time, setup and hold slack analysis, I/O analysis, recovery and removal, timing models)

- **Timing Reports**
  - Reporting in Quartus Prime Vs reporting in TimeQuest
  - Custom, summary and diagnostic reports
  - Clock transfer, datasheet, Fmax reports
  - Slack histogram report
  - Detailed slack/path report, further path analysis

- **Introduction to Timing Constraints**
  - Importance of constraining
  - Enter constraints
  - SDC netlist terminology
  - Collections

- **SDC Timing Constraints for Clocks**
  - Internal and virtual clocks
  - Generated clocks (inverted clocks, phase shifted clocks
  - PLL clocks and derive_pll_clocks Altera SDC extension
  - Automatic clock detection and creation
  - Non ideal clock constraints (Jitter, latency on PCB)
  - Common clock path pessimism removal
  - Checking clock constraints
  - Report clocks

Day #7 Labs

- **Lab #1: TimeQuest Interface & Timing Reports Generation**
  - Start the TimeQuest GUI and create timing netlist for analysis
  - Use TimeQuest reports to verify that design meets timing
  - Use the SDC file to guide the Quartus Prime fitter

- **Lab #2: Timing Analysis: Clock Constraints**
  - Create SDC file for a given design
  - Create base and generated clock constraints
  - Analyze the clock constrained design

Day #8

- **SDC Timing Constraints for I/O**
  - Combinational I/O interface constraints (max & min delay constraints)
  - Constraining synchronous I/O
  - I/O timing: virtual clocks
  - Synchronous inputs constraints (setup and hold time calculations, set_input_delay max & min)
  - Synchronous outputs constraints (set_output_delay min& max, when to use each constraint)
  - Constrain I/O using Tcl variables
  - Checking I/O constraints (report SDC, report unconstrained path, report ignored constraint)

- **Asynchronous Paths**
  - TimeQuest & asynchronous ports
  - Recovery and removal
  - Externally registered
  - Internally registered
  - Checking asynchronous control constraints
  - What about truly asynchronous control inputs?

- **Timing Exceptions**
    - Timing exceptions types
    - False paths
    - set_clock_groups command
    - Verifying false paths and groups
    - Report exceptions
    - Multicycle types (setup, hold, end, start)
    - Case 1: opening the window
    - Case 2: shifting the window
    - Determining and applying multicycles
    - Multicycle path constraints
    - Reporting multicycles in TimeQuest
    - Exception priorities

<u>Day #8 Labs</u>

❖ **Lab #1: Timing Analysis: Synchronous I/O Constraints**
- Constrain synchronous input paths using SDC
- Constrain synchronous output paths using SDC
- Generate reports and analyze results

❖ **Lab #2: Timing Analysis: Timing Exceptions & Analysis**
- Constrain asynchronous input signals
- Eliminate timing violations by using timing exceptions

Day #9

- **Introduction to Synthesis**
  - The synthesis process
  - Designing for synthesis
  - Hardware inference
  - Inference vs instantiation
  - Simulation vs synthesis
  - Synthesizable and non-synthesizable VHDL constructs

- **Concurrent Signal Assignment Synthesis**
  - Inference from declarations
  - Integers vs standard logic arrays
  - Inference from 'Z' value
  - Inference from simple concurrent assignment statements
  - RTL & Technology map viewers
  - Quartus Prime synthesis attributes
  - Closed feedback loop assignment
  - Inference from when-else statement
  - Inference from unaffected keyword
  - Inference from don't care
  - Using std_match function
  - Inference from selected signal assignment statements
  - Realization of arithmetic & relational operators
  - Synthesis tips and guidelines

- **Sequential Statement Synthesis**
  - Simple assignment statements
  - Inference from if-then-else, if-then-elsif statements
  - Inference from case statements
  - Inference from loop statements

Day #9 Labs

- **Lab #1: Design Gray Code Incrementor**
  - Implement binary to gray code and vice versa algorithms for any number of bits
  - Synthesize the design and verify functionality

- **Lab #2: Design Programmable Priority Encoder**
  - Implement programmable priority encoder according to a given algorithm
  - Synthesize and verify functionality vs golden reference design

Day #10

- **Sequential Statement Synthesis "Deep Dive"**
  - Incomplete sensitivity list
  - Inference using signals vs variables
  - Latch vs flip-flop inference
  - Wait statements synthesis
  - Finite state machine synthesis
  - Quartus Prime state machine viewer
  - State machine coding style for synthesis
  - State machine encoding styles in Quartus Prime
  - Quartus Prime synthesizer attributes for state machines
  - Safe state machine

- **Inferring Common Logic Functions**
  - Quartus Prime VHDL templates
  - DFF with secondary control signals
  - Incorrect control signal priority
  - Shift registers in logic and RAM
  - Counters
  - Single port RAM
  - Dual port , single clock RAM
  - Dual port, dual clock RAM

- Initializing memory contents using files
- Unsupported control signals for RAM
- ROM

Day #10 Labs

❖ Lab #1: Priority Sorter

- Develop an algorithm that receives 8 vectors of 8 bits each, plus 8-bit vector of priorities
- Clock, reset and enable are also inputs to the circuit
- The output is 8 bits
- Each rising edge of the clock one vector from the 8 input vectors is chosen and sent to the output if it has the highest priority until all 8 vectors are output
- The priority vector first sort the ‘1’ positions and then the ‘0’ positions, so for example if the priority vector input is, “11000101” then the output every clock will be: vector number 7, then 6, then 2, then 0, and then 5,4,3,2 (‘1’ first then ‘0’)
- Synthesize the design
- Add timing constraint and verify timing and functionality
- Repeat the exercise using ROM as a container for the input vectors and priority vector

Day #11

- **In-System Debug**
  - In-System debug challenges
  - Planning
  - Techniques
    - Use of pins for debug (SignalProbe)
    - Internal logic analyzer (SignalTap II)
    - Use of debug logic
    - External logic analyzer (LAI)
    - Editing memory content (ISMCE)
    - Use of soft processor for debug (Nios II)
  - Use scenarios
    - Power-up debug
    - Debug of transceiver interfaces
    - Reporting of system performance
    - Debug of soft processors
    - Device programming issues
  - In-System debug checklist

- **SignalTap II in Details**
  - Design flow using the SignalTap II Logic Analyzer
  - Define trigger conditions
  - View, analyze, and use captured data
  - Embedding multiple analyzers in one FPGA
  - Configure the SignalTap II logic analyzer
  - Adding FSM state encoding registers
  - Specifying the sample depth
  - Capturing the data to a specific RAM type
  - Choosing the buffer acquisition mode (non-segmented versus segmented buffer)
  - Using the storage qualifier feature
  - Creating basic and advanced triggers
  - Creating a power-up trigger
  - Using external triggers
  - Using incremental compilation with the SignalTap II
  - Performance and resource considerations
  - Run the SignalTap II view, analyze, and used captured data

- Using MATLAB MEX function to capture data
- Remote debugging using the SignalTap

Day #11 Labs

- **Lab #1: Debugging with SignalTap II Embedded Logic Analyzer**
  - Create SignalTap II and compile with a given design
  - Configure the device with the SignalTap II instance
  - Use SignalTap II triggers to determine the problem with a design

- **Lab #2: advanced SignalTap II Usage**
  - Experiment with storage qualification
  - Experience with a state-based trigger
  - Enable an additional instance of the SignalTap II

Day #12

- **Putting it All Together**

In the final project, the participant gets an almost full design in VHDL along with a description of each block functionality.

The design consist of various blocks including:

- Reset synchronizer
- LFSR counter
- ROM
- Address generator
- Two Matrix
- Absolute Matrix
- Maximum Matrix
- PWM

The participant needs to do the following:

1. Design the missing blocks in VHDL
2. Write a testbench for the design, simulate it and fix any bug that doesn't match the specifications
3. Synthesize the design in Quartus Prime and verify that you get the required logic
4. Use TimeQuest to constrain the design with given clock frequency, and I/O delays
5. Place & Route the design and analyze timing by generating various reports. Fix any timing issues.
6. Verify the design on board with SignalTAP II

Part III – Optimization Package (3 Days)

Day #13

- **Multiple Clock Domains**
  - Why synchronous design?
  - Synchronization circuits introduction
  - Setup and Hold time violations
  - Metastability effects
  - MTBF formula
  - Metastability problem
  - Unique characteristics of MTBF

- **Synchronizers**
  - Synchronizer definition
  - Two FF synchronizer
  - Three FF synchronizer
  - Not recommended synchronization circuit
  - Proper use of a synchronizer
  - Unregistered signals sent across a CDC boundary
  - Registered signals sent across a CDC boundary
  - Passing a fast control signal
  - Wide enable signal detection
  - Narrow enable signal regeneration
  - Level alternation scheme
  - Synchronizing fast control signals into slow clock domains
  - Sampling long CDC pulse
  - Open loop solution and considerations
  - Closed loop solution and considerations
  - Passing multiple signals between clock domains
  - Capturing a bus example
  - Passing multiple control signals between clock domains
  - Synchronized pulse generation logic
  - Send-receive toggle-pulse generation
  - Multicycle path and FSM solutions
  - MCP with feedback
  - MCP with acknowledge feedback
  - Asynchronous FIFO
  - FIFO pointers implemented as binary counters vs gray code counters
  - Gray code incrementor design for high speed
  - 1-deep 2-register FIFO synchronizer
  - Design tips

- **Design Partitioning for Synchronization**
  - Synthesis of a multiple clock system
  - Where to synchronize?
  - Guidelines for design partitioning
  - Partitioning with multi-cycle path

- **Reset Synchronizers**
  - Synchronous and asynchronous reset differences
  - When to use synchronous and asynchronous reset
  - Asynchronous reset problem
  - Reset synchronizer
  - Non-coordinated reset removal
  - Sequenced coordination of reset removal

- **Metastability Analysis in Quartus Prime**
  - Managing metastability with Quartus Prime
  - Metastability analysis in Quartus Prime
  - Identifying synchronizers for metastability analysis
  - Timing constraints & metastability analysis
  - Metastability & MTBF reporting
  - Design example & analysis
  - MTBF reporting in TimeQuest
  - Synchronizer data toggle rate in MTBF calculation
  - False path reporting in TimeQuest
  - MTBF optimization
  - Controlling MTBF optimization

<u>Day #13 Labs</u>

❖ **Lab #1: Design Handshake Synchronization Protocol**
  - Implement the handshake protocol in VHDL according to a given specification
  - Simulate your design
  - Compile the design in Quartus Prime
  - Analyze MTBF
  - Optimize MTBF with Quartus Prime optimization techniques

Day #14

- **Introduction to high productivity**
  - Project Management Phases
  - Project goals challenges
    - Resources utilization
    - I/O timing
    - Inter-chip timing
    - Debug
    - Power consumption
    - Reuse
    - Third party IP
    - Engineering resources
    - Communication between teams
    - Design tools version
  - Reverse engineering and security

- **Understanding Resource Utilization Report**
  - Synthesis resource utilization report
  - Utilization by entity report
  - Fitter resource utilization report
  - Resource utilization in terms of ALMs
    - ALMs needed
    - ALMs used in final placement
    - Estimate of ALMs recoverable by dense packing
    - Estimate of ALMs unavailable
    - ALMs used for memory
  - Resource utilization challenges
  - Resource utilization optimization reports
    - Registers removed during synthesis
    - General register statistics
    - Inverted register statistics
    - Synthesis attributes effect on optimization reports
  - Resource utilization use cases
  - Resource optimization advisor
  - I/O assignment analysis
  - Using I/O Flip Flops

- Using useioff attribute
- Optimize source code guidelines
- Optimize synthesis for area globally
- Optimize synthesis for area using the assignment editor
- When to use Restructure multiplexers optimization
- When to use Register packing optimization
- Preserve register and keep logic attributes
- Maximum fanout attribute
- Remove fitter constraints
- Viewing routing congestion
- Flatten the hierarchy during synthesis
- Retargeting memory blocks
- How to use efficiently physical synthesis to reduce area
- Retargeting/balancing DSP blocks
- Limiting the number of DSP blocks
- FSM processing and safe FSM

- **Designing for Best Area Utilization**
  - Derivation of efficient HDL description
  - Resource sharing definition
  - Operator sharing
  - Operator sharing examples
  - Operator sharing in Quartus Prime
  - Automatic operator sharing limitation
  - Balancing operators
  - Multipliers balancing
  - Counter efficient design
  - Functionality sharing definition
  - Functionality sharing examples

- **Reducing Compilation Time**
  - Compilation time challenges
  - Compilation time advisor
  - Rapid recompilation
  - Smart compilation
  - Parallel compilation with multicore host
  - Incremental compilation concept
  - Tips to reduce synthesis time
  - Tips to reduce placement time

- Tips to reduce routing time
- Tips to reduce static timing analysis time

Day #14 Labs

- **Lab #1 : Apply resource optimization techniques to achieve the smallest design area**
- **Lab #2 : Apply Synthesis Attributes to Control Synthesis Results and Reduce Compilation Time**

Day #15

- **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

- **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

Day #15 Labs

- **LAB #1: Use pipeline technique to increase complex circuit performance**
- **LAB #2: Optimize arithmetic circuit performance**