Intel® FPGA Technical Training

Performance Optimization with Stratix 10 HyperFlex Architecture

Course Description

Intel Stratix 10 introduces its new architecture, HyperFlex along with 14nm Intel Tri-Gate technology.

The new architecture allows FPGA designers to control their design performance by using registers throughout the core fabric, which provides a better approach to retiming, pipelining and optimization, in order to achieve timing closure faster.

This course provides all practical know-how needed to achieve higher productivity in Intel Stratix 10 FPGAs.

The course goes deep into the HyperFlex architecture and expose all Quartus Prime Pro features, design new methodologies and new tools such as Fast Forward Compile.

The course provides practical tools and design methods for engineers in order to increase their productivity by finish their projects quicker with best results.

Course Duration

3 Days

Goals

1. Become familiar with Intel Stratix 10 HyperFlex architecture
2. Use Quartus Prime Pro features to take advantage of Stratix 10 HyperFlex architecture
3. Speculate design performance gains using Fast Forward Compile feature
4. Employ Hyper-Retiming strategy to improve design performance
5. Employ Hyper-Pipelining strategy to improve design performance
6. Employ Hyper-Optimization strategy to reach the highest performance

Intended Users

Digital hardware engineers and FPGA team leaders, who would like to design with Stratix 10 FPGA and achieve higher productivity in their design

Previous Knowledge

Quartus Prime Pro software

VHDL/Verilog/SystemVerilog

TimeQuest

FPGA design experience

Course Material

1. Quartus Prime Pro
2. Modelsim
3. Course book (including labs)

Table of Contents

- **Hyper-Retiming**
  - Why the Stratix 10 architecture is fast?
  - Retiming as an optimization tool
  - “Conventional” register Retiming in the Quarus Prime software
  - Conventional register retiming challenges
  - What is Hyper-Retiming?
  - Performance gain with Hyper-Retiming
  - Hyper-Retiming advantages
  - How does Hyper-Retiming work?
  - Hyper-Retiming decision making
  - Solver constraints
  - The critical chain: when there is no solution
  - Converging and diverging paths
  - Retiming dependency rules
  - The Retiming dependency node
  - Hyper-Retiming complexity
  - Registers that cannot be Retimed
  - Path endpoint restrictions
  - Common design situations restricting register Retiming
    - Asynchronous clears
    - Synchronizer chains
    - I/O boundaries
    - Clock crossings
    - SDC timing exceptions
    - User “preserve” synthesis attribute
    - Dual-clock RAMs and DSPs
    - Unpreserved initial conditions
    - Latches
  - Performance improvement vs. design effort
  - Fast Forward Compile and Hyper-Retiming
  - Settings for controlling Fast Forward Compile – Retiming restrictions
  - Hyper-Retiming tools

  ❖ **Lab #2 : Hyper -Retiming**

- **Hyper-Pipelining**
  - General benefits of heavily pipelined designs
  - Conventional pipelining
  - Performance gain with conventional pipelining
  - What is Hyper-Pipelining?
  - Performance gain with Hyper-Pipelining
  - Path endpoint restrictions
  - Endpoint restricted path fixed by Hyper-Pipelining
  - Hyper-Pipelining dilemma
  - Hyper-Pipelining – the "easy way"
  - Fast Forward Compile for Hyper-Pipelining
  - Where will registers be added?
  - Controlling Fast Forward Compile – maximum pipelining
  - Data synchronization and Fast Forward Compile
  - Pipeline registers vs. synchronizer chain identification
  - Why seed sweep?
  - Design Space Explorer II
  - Hyper-Pipelining considerations

❖ **Lab#3: Hyper-Pipelining**

- **Introduction to Hyper-Optimization**
  - Hyper-Optimization vs. Hyper-Retiming/Hyper-Pipelining
  - What is Hyper-Optimization?
  - Non-optimized feedback loop
  - Optimized feedback loop
  - Why are loops barriers to Hyper-Retiming?
  - Retiming a loop
  - Illegal loop retiming
  - Loops with multiple registers
  - Why are loops such a big deal now?
  - Hyper-Optimization notes
  - Questions to think about when rearchitecting

- **Evaluating Hyper-Optimization**
  - Stratix 10 Retiming limit
  - The critical chain
  - Critical chain analysis

- Critical chain report tabs
- Critical chain details example
- Retiming dependency
- Recommendations for critical chain
- Critical chain types
- Insufficient registers critical chain
- Fixing insufficient registers with Hyper-Pipelining
- Fixing overlapping insufficient registers with Hyper-Pipelining
- Path limit critical chain
- Short path/Long path critical chain
- Fixing Short path/Long path critical chain types
- Short path/Long path critical chain example
- Fixing Short path/Long path critical chain by duplicating nodes
- Loop critical chain
- Retiming dependencies and loops
- "Seeing" the loop with Retiming dependencies
- Simple RTL loop critical chain example
- More complicated Retiming dependency loop
- Critical chain bus trick
- Fixing loop critical chain

- **Fast Forward Compile for Hyper-Optimization**
  - Fast Forward Compile DSP/RAM block analysis
  - Example Fast Forward report
  - Controlling Fast Forward Compile RAM/DSP Hyper-Optimization
  - Use Fast Forward limit for maximum performance
  - Avoid overzealousness!
  - Identify loops using Fast Forward Compile critical chains
  - Methods to identify/locating loop
  - Loop vs. Short path/Long path reasons

- **Loop Examples & Solutions**
  - Loop analysis
  - Loops due to resets
  - Loops due to counters
  - Loops due to state machines
  - Loops due to algorithm
  - Loops due to hardware reuse

- Loops due to FIFO signals
- Additional FIFO issues
- Loops in particular to watch out for
- Unroll loops
- Modify FIFO threshold
- FIFO skid buffers
- How does the skid buffer work?
- Hyper-Optimization with FIFO skid buffers
- Feed forward flow control with data valid pipeline
- Feed forward flow control with pipelined feedback
- Applying loop solutions
- “Unrecognized” state machine
- Advanced Hyper-Optimization

❖ **Lab#4: Loop Unrolling**

- **Pre-Computation Techniques**
  - Pre-computing to reduce loop size
  - Removing work from feedback loops
  - Pre-computation counter example
  - Pre-computation accumulator example
  - Reduce loop signal dependency
  - Pre-computation packet length processing example
  - Pre-computation on state machines
  - Modulus 24 example
  - Level 2 Ethernet Backhaul Switch case study

- **Shannon’s Decomposition**
  - Shannon’s decomposition in hardware
  - Synthesizing Shannon’s decomposition
  - Expanding Shannon’s decomposition
  - Shannon’s decomposition on loops
  - Loop retiming
  - Generalizing Shannon’s decomposition on loops
  - Evaluating Shannon’s decomposition
  - Pipelined feedback loop example
  - Shannon’s decomposition Toolbox
  - Toolbox example
  - Complex loop toolbox example

- Three loops “Shannonized”
- Shannon’s decomposition in practice
- Controlling synthesis when optimizing with Shannon’s
- Notes on Shannon’s decomposition on state machines
- Shannon’s decomposition case study

❖ **Lab#5: Shannon’s Decomposition**

- **Manual Loop Acceleration Techniques**
  - TDM Retiming to speed up loops
  - When to use TDM Retiming
  - Hyper-Folding introduction
  - Performing Hyper-Folding
  - Loop pipelining introduction
  - Loop pipelining function f()
  - Adder/accumulator example
  - IIR filter example
  - Applying loop pipelining with cascading loop logic
  - BCH encoder case study
  - Loop pipelining and multi-threading partial sums/products/states

❖ **Lab#6: Hyper-Folding (Manual Loop Acceleration)**