Intel Hyperflex
Reconfigurable Computing Architecture
A Thomas Cenova and Christopher Sweet presentation
Table of Contents

- Introduction
- Performance Increases
  - Hyper-Retiming
  - Hyper-Pipelining
  - Hyper-Optimization
- Power & Size Considerations
- Conclusion
Table of Contents

- Introduction
- Performance Increases
  - Hyper-Retiming
  - Hyper-Pipelining
  - Hyper-Optimization
- Power & Size Considerations
- Conclusion
Introduction - The Need for a New Architecture

- New data transport methods
  - Bus sizes ranging from 512 bits to 2048 bits.
  - Larger bus sizes limit maximum frequency
- Advanced processing nodes
- Localized Array Blocks (LABs)
- Pipelining Quandary
- Clock Trees and Skew
Introduction - Intel HyperFlex

- Redesigned DSP Blocks and M2DK Memory Blocks
- Programmable Clock Tree Synthesis
- Hyper Registers
  - Interconnects throughout FPGA
  - Adaptive Logic Modules (ALMs) and Functional Registers
- Three methods of design improvement
- Intel Stratix 10 fpga implement HyperFlex
  - Samples out beginning in October 2017

Figure 1: Hyper Register Interconnects
Table of Contents

- Introduction
- Performance Increases
  - Hyper-Retiming
  - Hyper-Pipelining
  - Hyper-Optimization
- Power & Size Considerations
- Conclusion
Hyper-Retiming - Conventional Retiming

- Find unused ALMs and include them in the circuit
- Incurs delays if:
  - ALM has poor locality
  - Large bus width
  - Algorithmic complexity

**Figure 2:** (Right) - Retiming after the addition of an unused ALM
Hyper-Retiming - The HyperFlex Way

- Remove Registers out of logic cells
  - Shifts focus to interconnections
  - Delay of a Hyper Register
  - No impact on existing architecture
  - Little to no user effort to implement
  - Average 1.5X speedup

**Figure 3:** (Right) - Hyper-Retiming allows for even faster clock speeds when using the same ALM
Table of Contents

- Introduction
- Performance Increases
  - Hyper-Retiming
  - Hyper-Pipelining
  - Hyper-Optimization
- Power & Size Considerations
- Conclusion
Hyper-Pipelining - Standard Pipelining

- **Drawbacks of pipelining**
  - Slow, Iterative Process
  - Potentially ‘orphaned’ registers
  - Specific design structure

- **Advantages**
  - Even higher clock speeds!

*Figure 4: Pipelining through Hyper Registers*
Hyper-Pipelining - Similar but Different

- Cost free pipelining
- Less ALMs used
- Can be completed live / without iteration
- Minor user effort to implement
- Average 1.65X Speedup

Figure 4: Pipelining through Hyper Registers
Table of Contents

- Introduction
- Performance Increases
  - Hyper-Retiming
  - Hyper-Pipelining
  - Hyper-Optimization
- Power & Size Considerations
- Conclusion
Hyper-Optimization - Fast Lookup Values

- Optimizations in design take many forms
  - Commonly involves precomputing potential values and using a LUT
  - Hyper Registers only increase the speed of the look-up operation

**Figure 5:** (Right) - Loop conversion to Lookup Operation for optimal design
Hyper-Optimization - Performance Gains

- Requires more user effort to implement and is dependent on design
  - Logic sections need to use feed-forward or precompute paths
  - Designs should not use long combinatorial feedback paths to maximize speedup [3]
- Average 2.0X speedup
Table of Contents

- Introduction
- Performance Increases
  - Hyper-Retiming
  - Hyper-Pipelining
  - Hyper-Optimization
- Power & Size Considerations
- Conclusion
Power & Size Considerations

- Conventional FPGAs
  - Global Clock Tree
    - High Fan-out
    - Chip-wide Clock Domain
  - Rate Matching / Clock Crossing subpar
- HyperFlex Way:
  - Pre-routed clock paths (ASIC Design)
  - Localized clock domains
  - Optimal Clock Skew Behavior
- Power
  - 14nm FinFET Technology
  - Smaller, High Clock speed designs save on static power

Figure 6: Clock tree synthesis may be balanced or unbalanced as need
Hyper-Folding

- Hyper-Folding is the reduction of a design's implementation size and power while also running at twice the core frequency.
- Reduction of power comes from 2 things:
  - Fewer device resources results in less static power.
  - Fewer device resources results in lower dynamic power when increasing the clock frequency.  
    - Power Dynamic = ½ * C * V^2 * F
    - Less logic resources lowers C, resulting in less overall dynamic power.
- Core frequency improvements come from Hyper-Flex optimizations.
In Review

- Registers Everywhere approach
- Three forms of design optimization
- Enables future state of the art designs
- Questions?
References


