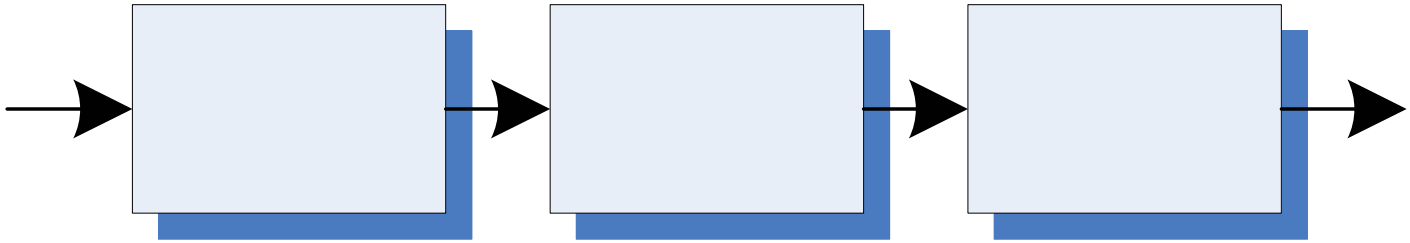
Actel®
POWER MATTERS

Physical optimization for FPGAs using post-placement topology rewriting

Val Pevzner, Andrew Kennings, Andy Fox

Introduction (1)

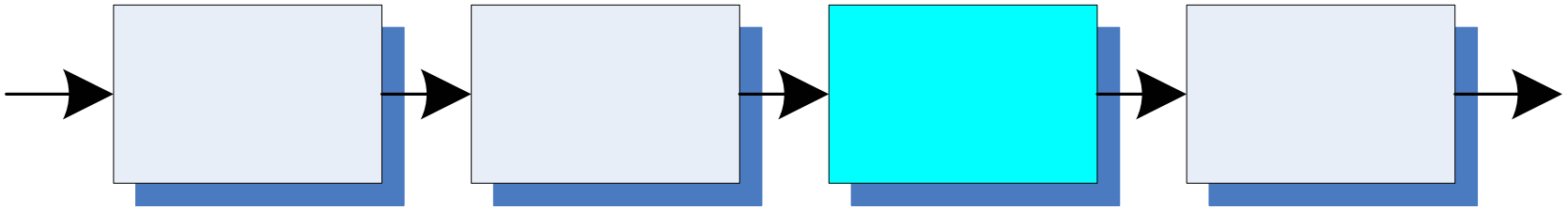
- Traditional flow for backend of FPGA tools:



- Many useful improvements made in each of these steps to address objectives of timing, area, power, etc…
- Typically understood, however, that:
  - **Placement and routing are bound by the output of technology mapping; and**
  - **Technology mapping is potentially forced to work with inaccurate information with respect to delay.**

Introduction (2)

- Interconnect delay increasingly important for FPGA design and physical information is required!
- More typical/modern flow:



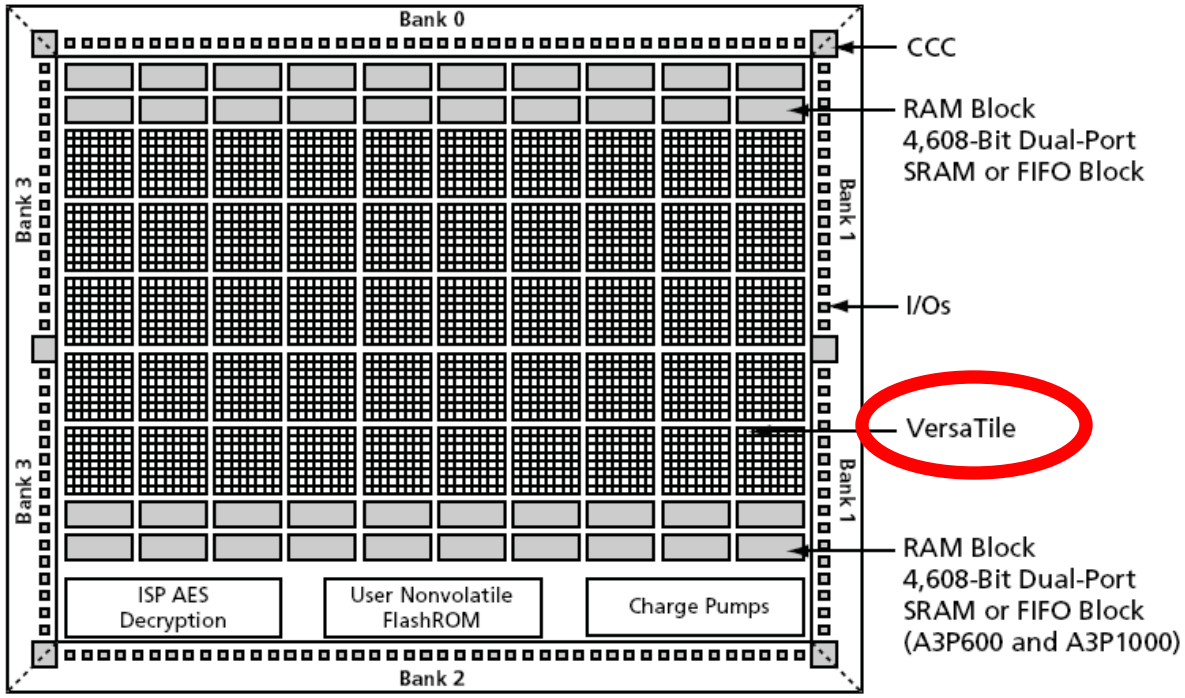
- Insertion of post-placement optimizations can significantly improve the ability to optimize design objectives.
- More accurate estimate of delay and likely interconnect is available.
- Should exploit physical information *AS WELL AS the particular architecture imposed by the FPGA being considered.*

Prior physical optimizations for FPGAs

- Different techniques proposed for FPGA post-placement optimizations:
  - **Logic duplication + empty resources [Schabas & Brown; 2003];**
  - **Logic duplication with feasible regions and monotonic paths + incremental placement [Beraudo & Lillis, 2003];**
  - **Shannon decomposition + incremental placement [Singh & Brown, 2007];**
  - **Timing-driven functional decomposition + incremental placement [Manohararajah, Singh & Brown, 2005];**
  - **Logic decomposition with choices and remapping + incremental placement [Kim & Lillis, 2008].**
- The different methods are all linked tightly with incremental placement (important) and rely on logic duplication and/or decomposition strategies.

ProASIC3 Architecture (1)

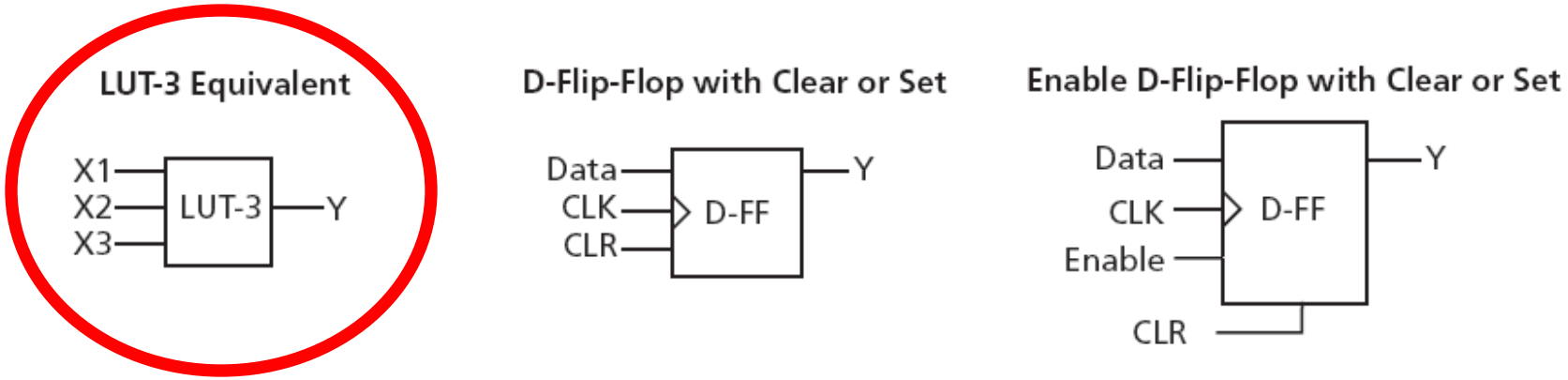
- Device level architecture of the Actel ProASIC3 (+related devices and families; Igloo, Nano, ...).



Source: ProASIC3 Handbook 2/2009; Figure 1.2

ProASIC3 Architecture (2)

- The VersaTile is capable of implementing both combinational and sequential logic.
- Need to exploit the feature of the architecture; namely the fact we are working with LUT3



Source: ProASIC3 Handbook 2/2009; Figure 1.3

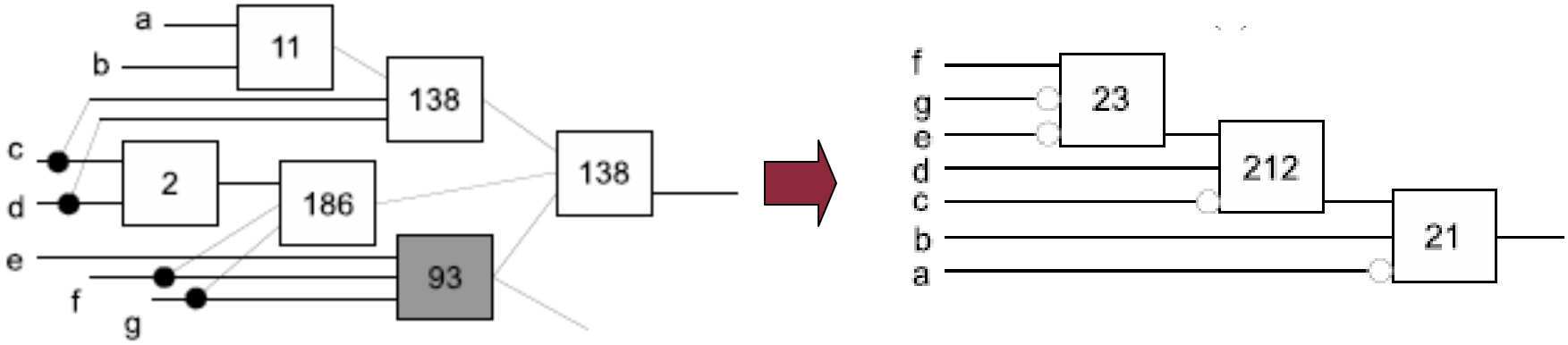
This Paper

- Our proposal is a post-placement optimization based on the concept of circuit rewriting with predefined circuit topologies.
  - **Conceptually very simple; similar to those methods used for AIG rewriting;**
  - **More powerful than pure logic duplication;**
  - **Abstracts out the requirements of any particular decomposition technique;**
  - **Tightly integrated with incremental placement to ensure accurate timing information.**
- Requires some off-line (a priori) processing to prepare the circuit topologies.
- Ability to perform the off-line processing (as we shall see) is a consequence of the FPGA architecture being considered (LUT3)!

Rewriting

- A cone of logic is selected and simulated. A comparison is made to a library of alternative circuit topologies capable of implemented the function.
  - **If the alternative implementation improves the result, then the original cone of logic is replaced or – *rewritten* – with the alternative implementation.**
  - **Iteratively applied either to all or a subset of nodes in a network, often in forward or reverse topological order.**
- For FPGA, typically applied prior to technology mapping to optimize an AIG.
- Assuming that it is possible to compute an alternative set of circuit topologies, the same concepts can be applied to a LUT graph.

Example of rewriting LUT



7-input cone of logic; cone consists of LUT2 and LUT3

7-input cone of logic implementing the same function.

- The rewrite will improve area (less LUT) and may improve timing (depending on placement, delays, etc.)

Top-level algorithm

- Effectively the same as any rewriting algorithm with appropriate modifications to account for selection of nodes to rewrite, incremental placement and incremental timing analysis.

```
Procedure: Post-placement rewriting
Input/Output: A placed LUT netlist N
begin
   Identify a set S of timing critical nodes in N via timing analysis;
   for each node n ∈ S do
      // Find set of ≤ k-input cones C of logic rooted at n
      compute_cones(c,k);
      for each c ∈ C for node n do
         Compute logic function f of c;
         // Compute set of alternatives LUT topologies M that implement f
         M = match_topology(f);
         for each m ∈ M do
            // Rewrite the k-input cone c with topology m
            rewrite_topology(c,m);
            // Perform incremental placement and timing analysis
            incremental_placement(c,m);
            incremental_timing_analysis();
            if ( timing_improved ) then
               // Implementing f with m better than with c;
               accept_topology(c,m);
               goto next_node;
            else
               reject_topology(c,m);
            end if
         end do
      end do
   next_node:
      ;
   end do
end
```

Select timing critical nodes

Consider different logic cones for each node

Find alternative LUT topologies for cone

Incremental placement and timing

Accept or reject current rewrite

Matching cones to LUT topologies

- Given pre-encoded topologies of LUT, functions of logic cones can be tested for feasibility very quickly using encoding (NPN) and hash lookups.

Procedure: match_topology(c, lib);
Input: k-input cut c and encoded topology library lib.
Output: true (match) or false (no match), set of matches and match details, M.
begin
  // determine the function implemented by the cut
  f ← simulate_cut(c);
  // determine the equivalence class for f
  g ← npn_encode(f, ip_perm, ip_phase, op_phase);
  // determine if topologies implementing g exists.
  bool *retval* ← lib::match(g);
  return *retval*;
end

simulation

encoding

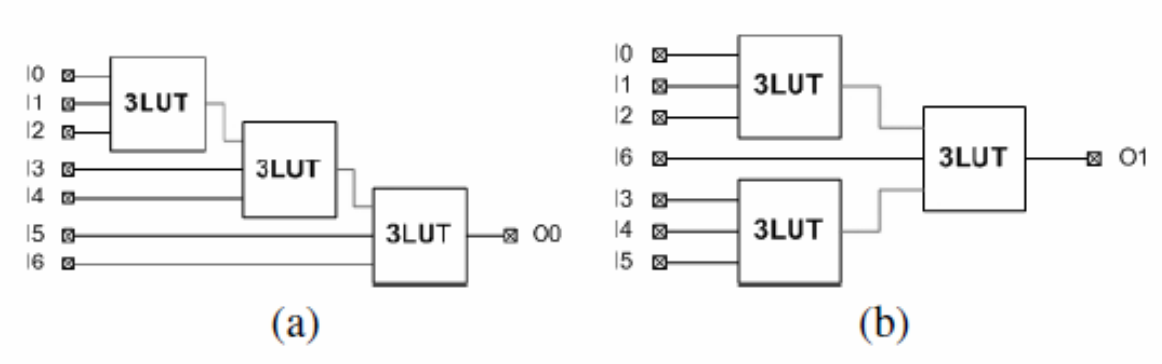
hash lookup

Topology Encoding (1)

- Must encode LUT topologies to facilitate fast matching.
  - **Matching logic functions to LUT topologies using SAT is great [Hu et al., 2007], but time consuming.**
- Can also consider using NPN encoding (a la cell libraries).
  - **For a given set of LUT topologies, determine all functions that each topology can implement;**
  - **Encode functions using NPN to reduce storage and matching times.**
  - **All this simulation and encoding is done a priori, off-line and information is stored in data files.**
- *The ability to encoding and matching is a result of the FPGA architecture under consideration!*
  - **Topologies consisting of LUT with <= 3 inputs are realistic to encode to a sufficient number of inputs (don't implement too many different functions!)**
  - **E.g., quite practical to get up to (and including) 9-input functions which proved to be sufficient.**

Topology Encoding (2)

- Samples topologies for 7-input functions:



- Off-line, a priori simulation and encoding:

Can exploit symmetry to skip many of the configuration bits (simulated functions lead to the same equivalence class).

Incremental placement

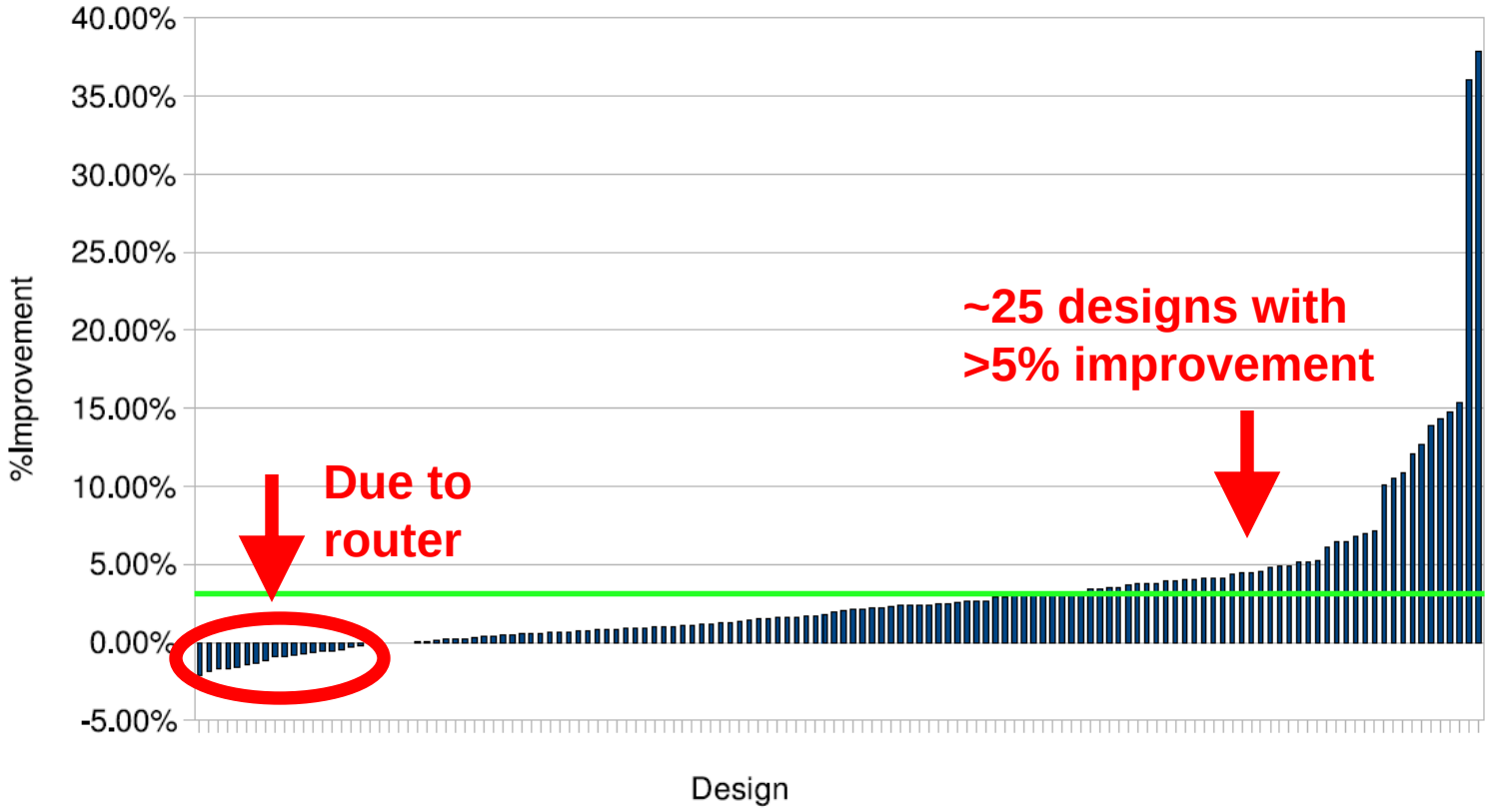
- After each rewrite, we need to perform both incremental placement and timing analysis.
  - **In FPGA, the incremental placement problem is very specific to the FPGA architecture being considered.**
- For ProASIC3, the incremental placement problem is relatively simple due to the flat homogeneous architecture of the device.
- Incremental placement method:
  - **Rip-up the LUT in the cone being rewritten (creates gaps in placement);**
  - **Place LUT from alternative topology into their feasible regions for monotonic paths;**
  - **Perform rippling to remove any overlaps.**

Numerical results (1)

- Algorithm implemented in C++ (within commercial tool flow).
- Used a small number of LUT3 topologies encoded off-line suitable for matching logic cones with up to 7-inputs.
- Tested rewriting algorithm on a set of 136 industrial design cases.

Numerical results (2)

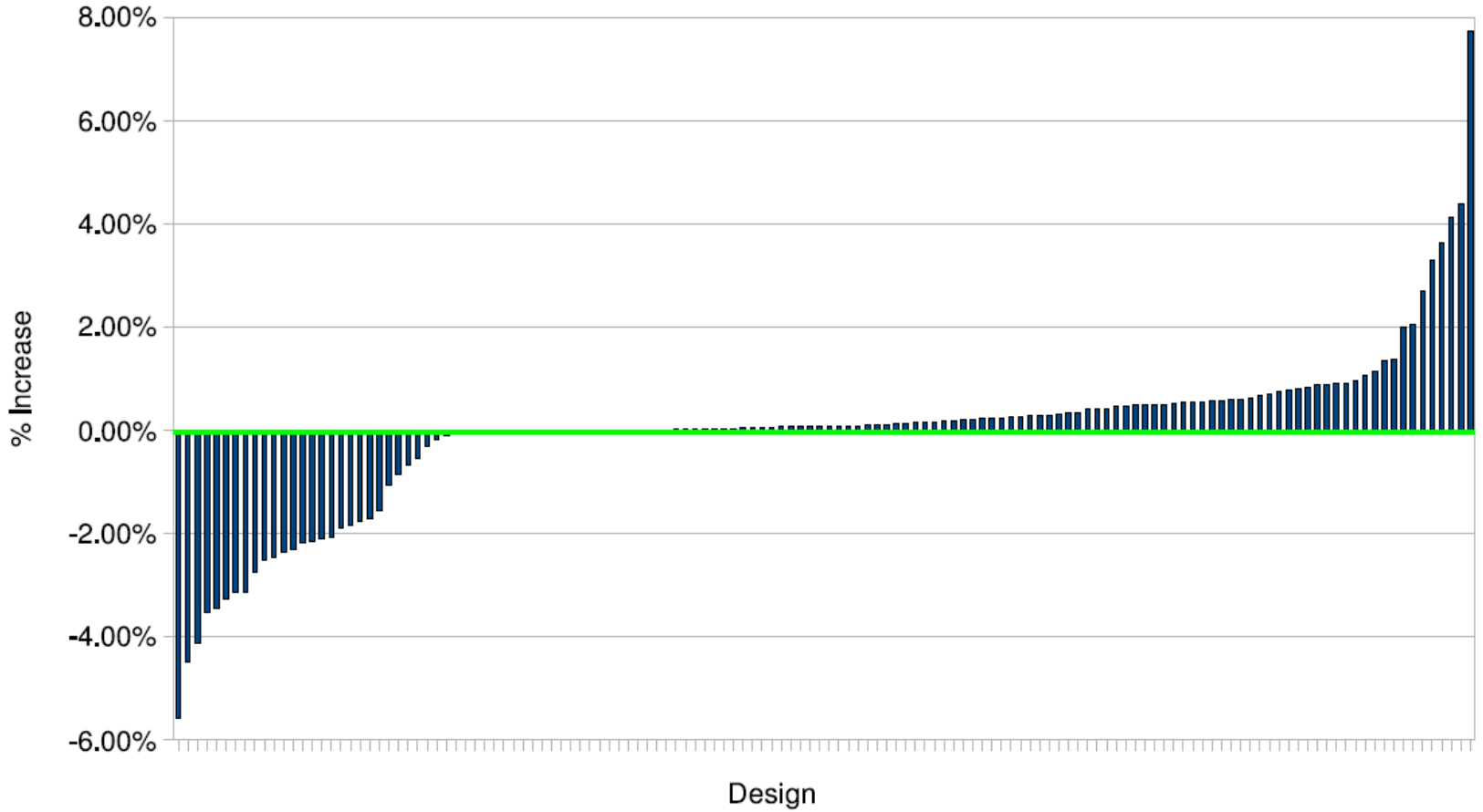
- Test#1: Percentage improvement in post-routed quality of result (timing performance; improvement in post-routed slack).



- Average improvement of ~ 3.1% with max. improvement of 37.9% *on top of existing physical optimization algorithms*.

Numerical results (3)

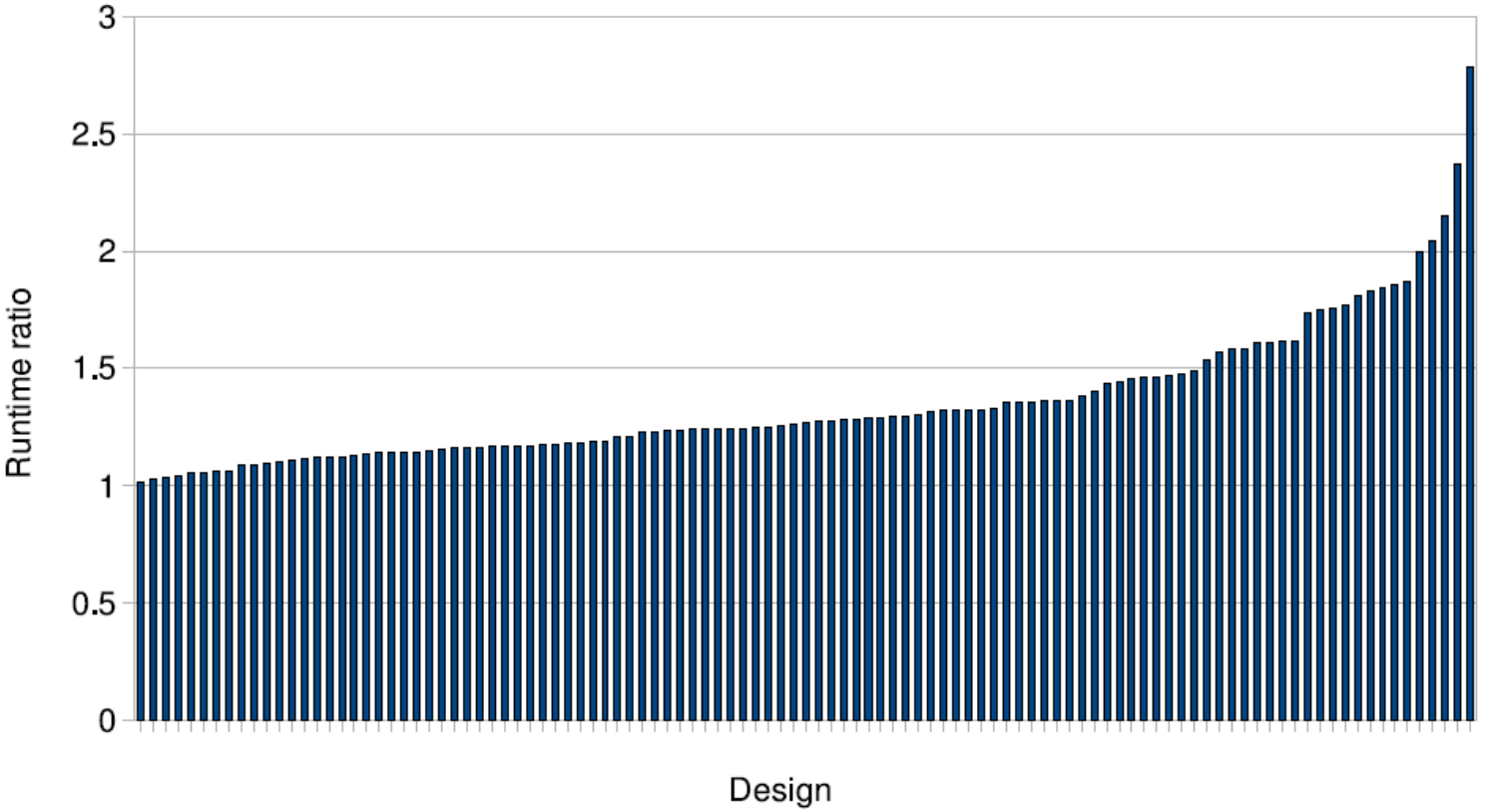
- Test#2: Impact on design area.



- On average, negligible impact on circuit area; circuit area is not an issue anyway (designs all fit; no power impact).

Numerical results (4)

- Test #3: Impact on run-time.



- Average of 1.4X larger run-time on designs that took >2 minutes. Increase in run-time is more a consequence of incremental placement and timing analysis; Not the encoding/matching steps!

Conclusions

- Presented a post-placement optimization algorithm for FPGA that relies on conceptually simple algorithm of circuit rewriting.
  - **Tightly integrated with incremental placement;**
  - **Targeted to a commercial FPGA architecture (ProASIC3);**
  - **Uses NPN encoding + matching to find alternative circuit structures; possible because the architecture is composed on LUT3.**
- Tested on an industrial suite of test circuits.
  - **Yielded a small improvement of ~ 3.1% over all designs, but as much as 37.9%.**
  - **Minor increase in design area (expected);**
  - **Increase in run-time (but due to the need for incremental placement and incremental timing analysis).**

Questions?