Straight to the Point: Intra- and Intercluster LUT Connections to Mitigate the Delay of Programmable Routing

S. Nikolić, G. Zgheib*, and P. Ienne

FPGA'20, Seaside, 24.02.2020

École Polytechnique Fédérale de Lausanne
*Intel Corporation

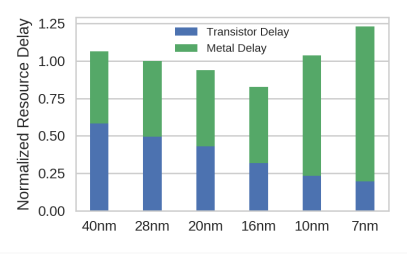
Interconnect Doesn't Scale Very Well...

Xilinx Adaptive Compute Acceleration Platform: Versal™ Architecture

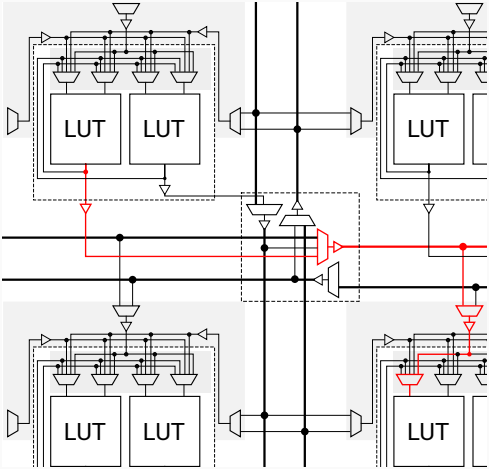
Brian Gaide, Dinesh Gaitonde, Chirag Ravishankar, Trevor Bauer
bgaide@xilinx.com,dineshg@xilinx.com,chiragr@xilinx.com,trevor@xilinx.com
Xilinx Inc.

ABSTRACT

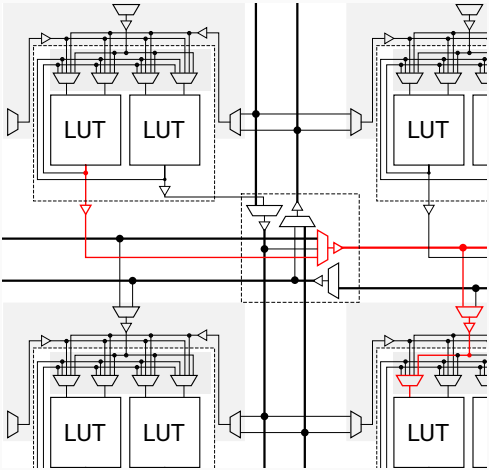
In this paper we describe Xilinx's Versal™ Adaptive Compute Acceleration Platform (ACAP). ACAP is a hybrid compute platform that tightly integrates traditional FPGA programmable fabric, software programmable processors and software programmable accelerator engines. ACAP improves over the programmability of traditional reconfigurable platforms by introducing newer compute models in the form of software programmable accelerators and by separating out the data movement architecture from the compute architecture. The Versal architecture includes a host of new capabilities, including a chip-pervasive programmable Network-on-Chip (NoC), Imux Registers, compute shell, more advanced SSIT, adaptive deskew of



A Typical Connection

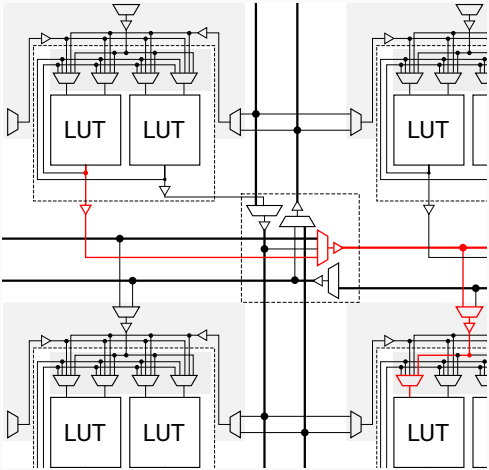


A Typical Connection



Quite a few transistors...

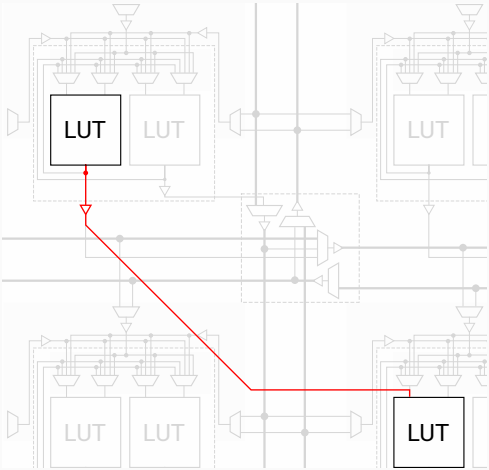
A Typical Connection



Quite a few transistors...

Start by removing (some of) them?

A Typical Connection

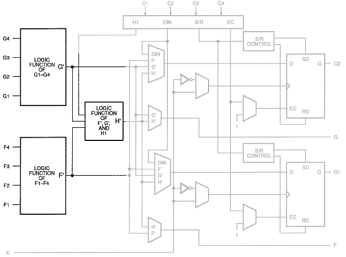


Quite a few transistors...

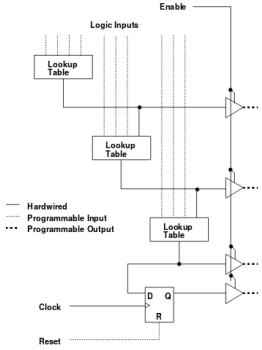
Start by removing (some of) them?

Back to the Future?

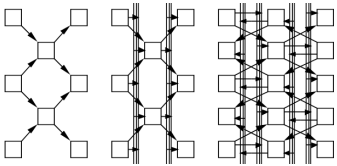
XC4000 [1]



UTFPGA1 [2]



Triptych [3]



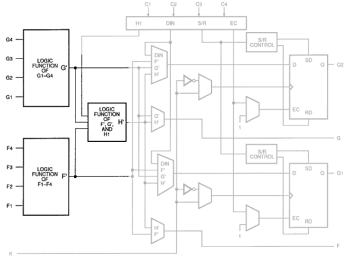
[1] H.-C. Hsieh, W. S. Carter, J. Ja, E. Cheung, S. Schreifels, C. Erickson, P. Freidin, L. Tinkey, and R. Kanazawa. Third-generation architecture boosts speed and density of field-programmable gate arrays, 1990

[2] P. Chow, S. O. Seo, D. Au, B. Fallah, C. Li, and J. Rose. A 1.2um CMOS FPGA using cascaded logic blocks and segmented routing, 1991

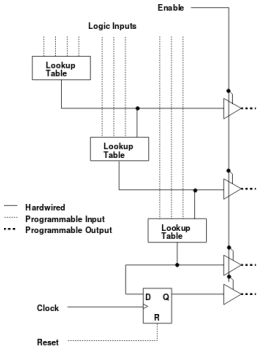
[3] C. Ebeling, G. Borriello, S. A. Hauck, D. Song, E. A. Walkup. TRIPTYCH: A New FPGA Architecture, 1991

Not So Fast...

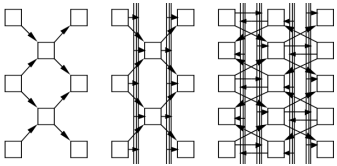
1990



1991



1991



[1] H.-C. Hsieh, W. S. Carter, J. Ja, E. Cheung, S. Schreifels, C. Erickson, P. Freidin, L. Tinkey, and R. Kanazawa. Third-generation architecture boosts speed and density of field-programmable gate arrays, 1990

[2] P. Chow, S. O. Seo, D. Au, B. Fallah, C. Li, and J. Rose. A 1.2um CMOS FPGA using cascaded logic blocks and segmented routing, 1991

[3] C. Ebeling, G. Borriello, S. A. Hauck, D. Song, E. A. Walkup. TRIPTYCH: A New FPGA Architecture, 1991

Not So Fast…

Directional and Single-Driver Wires in FPGA Interconnect

Guy Lemieux Edmund Lee Marvin Tom Anthony Yu
Department of ECE, University of British Columbia
Vancouver, BC, Canada

E-mail: { lemieux | eddy1 | marvint | anthonyy } @ ece.ubc.ca

FPT'04

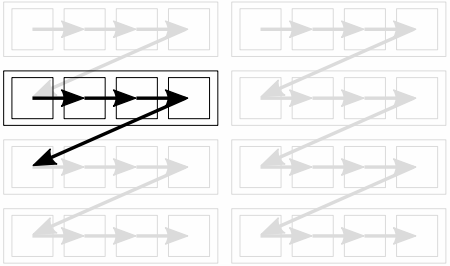
Using Cluster-Based Logic Blocks and Timing-Driven Packing to Improve FPGA Speed and Density

Alexander (Sandy) Marquardt, Vaughn Betz, and Jonathan Rose
Department of Electrical and Computer Engineering
University of Toronto
Toronto, ON, Canada M5S 3G4
{arm,vaughn,jayar}@eecg.toronto.edu

FPGA'99

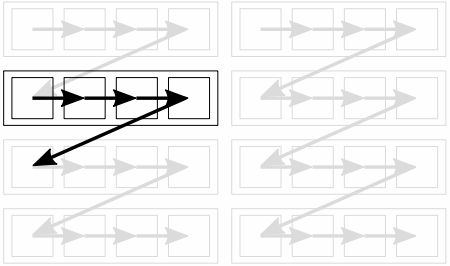
Not So Fast...

All these patterns are very simple

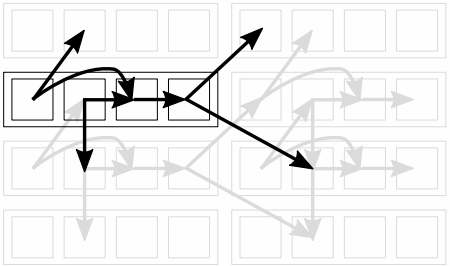


Not So Fast...

All these patterns are very simple



Do they really achieve all of the hardening potential?



In This Work

Does it make sense to harden complex patterns to reduce delay?

How should these patterns look like?

In This Work

Does it make sense to harden complex patterns to reduce delay?

- Yes, it does

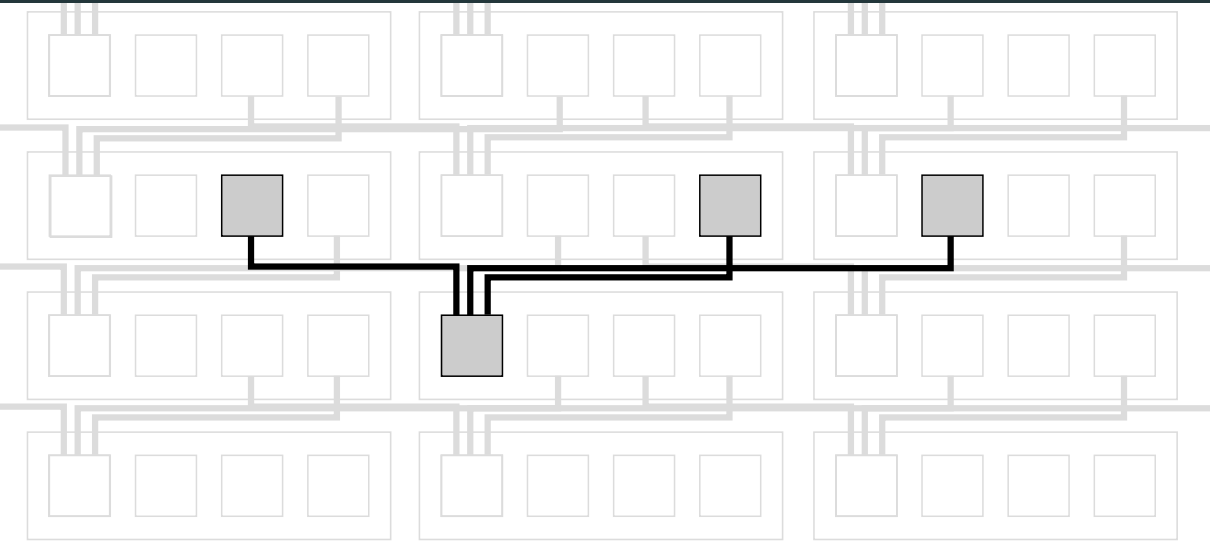
How should these patterns look like?

- We give an algorithm

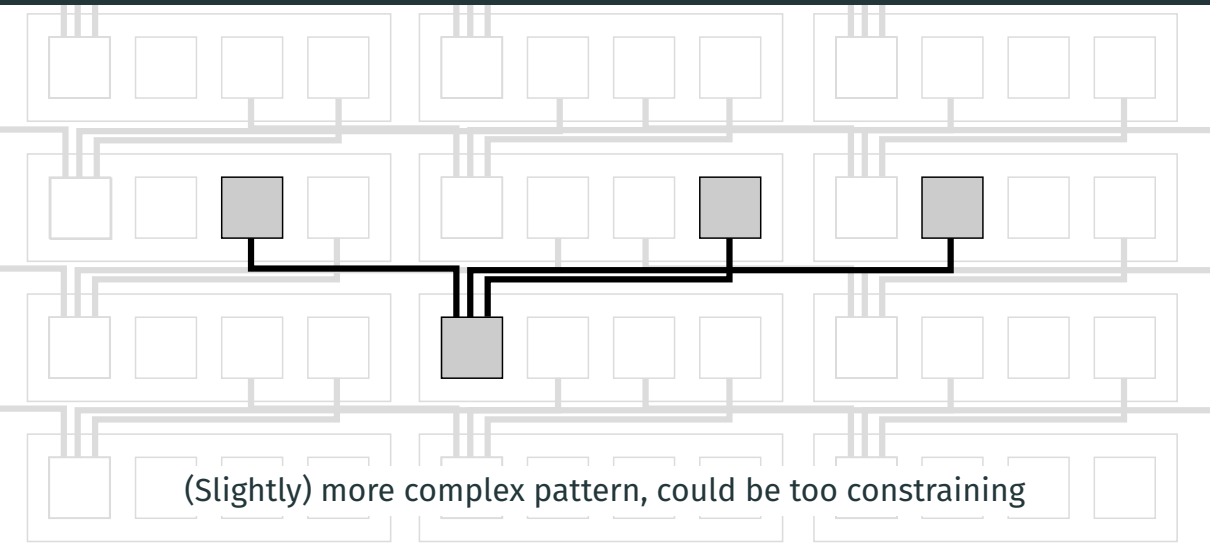
Outline

Which Patterns?

Issues With Full Hardening

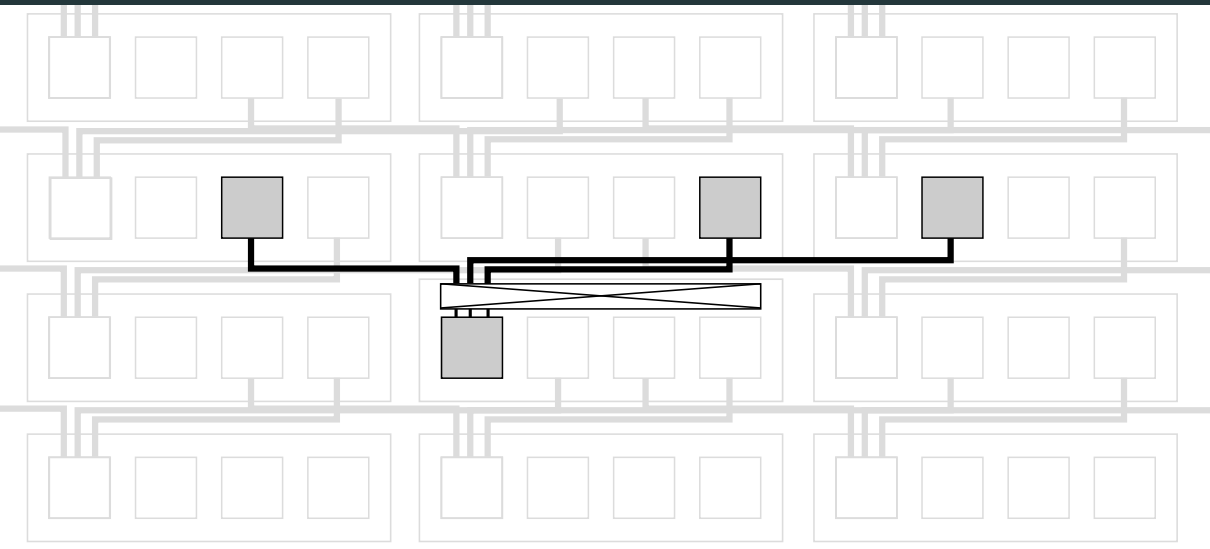


Issues With Full Hardening

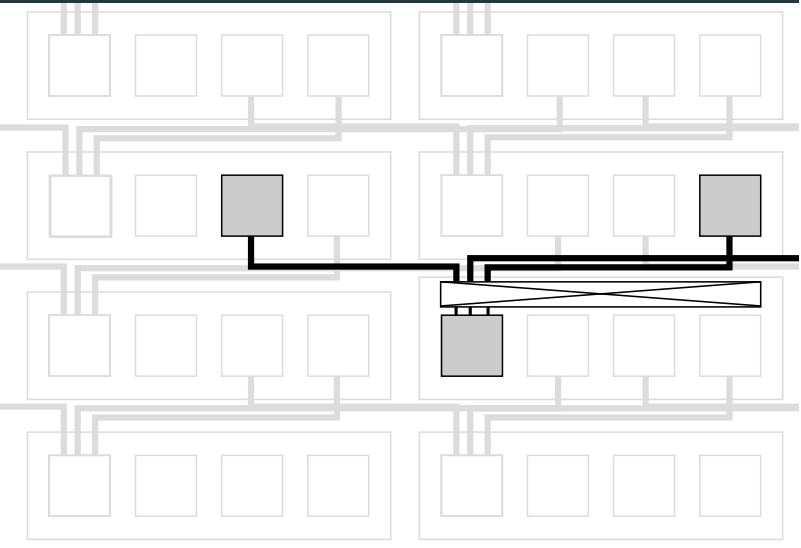


(Slightly) more complex pattern, could be too constraining

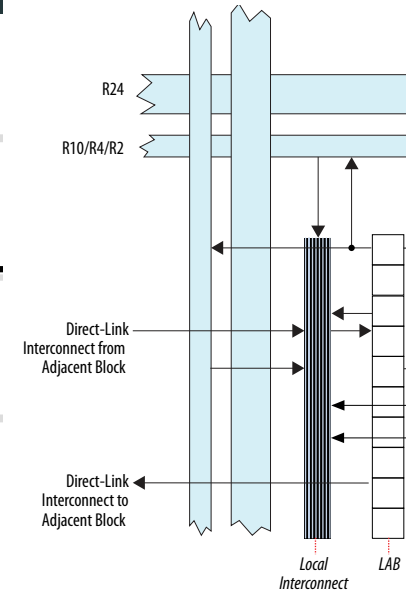
Issues With Full Hardening: A Solution?



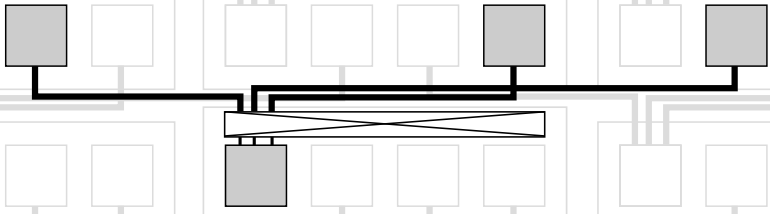
Issues With Full Hardening: A Solution?



Intel® Stratix® 10 Logic Array Blocks and Adaptive Logic Modules User Guide

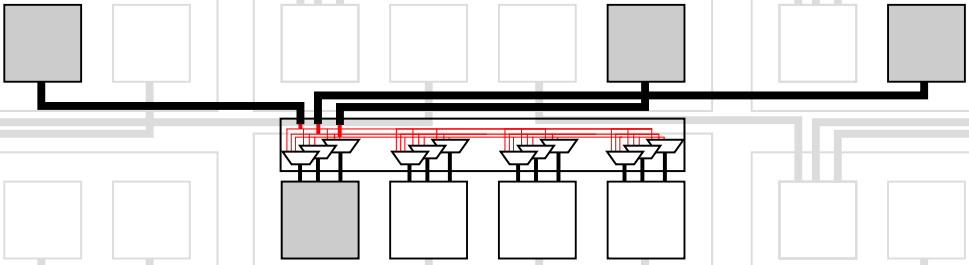


Issues With Full Hardening: A Solution?



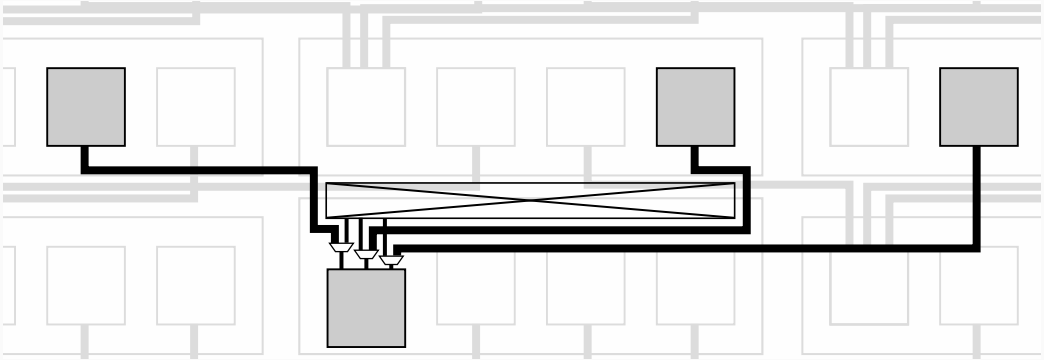
Delay of the crossbar not eliminated

Issues With Full Hardening: A Solution?



High cost of broadcasting

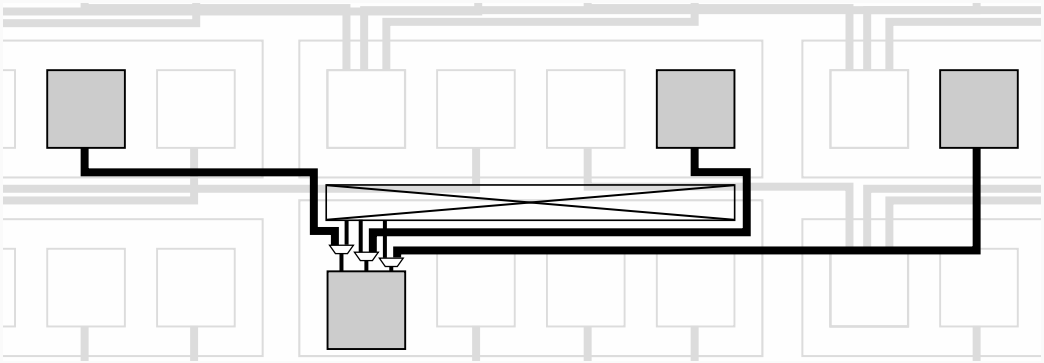
Issues With Full Hardening: A Compromise



Each direct connection is decoupled by a multiplexer

[1] X. Tang, P.-E. Gaillardon, G. De Micheli, “Pattern-based FPGA logic block and clustering algorithm”, FPL’14
[2] W. Feng, J. Greene, A. Mishchenko, “Improving FPGA Performance with a S44 LUT Structure”, FPGA’18
[3] B. Gaide, et al., “Xilinx Adaptive Compute Acceleration Platform: Versal™Architecture”, FPGA’19

Issues With Full Hardening: A Compromise



We use this approach

Which Patterns?

- All the programmable interconnect flexibility retained at a minimal cost

Which Patterns?

- All the programmable interconnect flexibility retained at a minimal cost
- No placement constraints

Which Patterns?

- All the programmable interconnect flexibility retained at a minimal cost
- No placement constraints
- All existing CAD tools still work (if suboptimally)

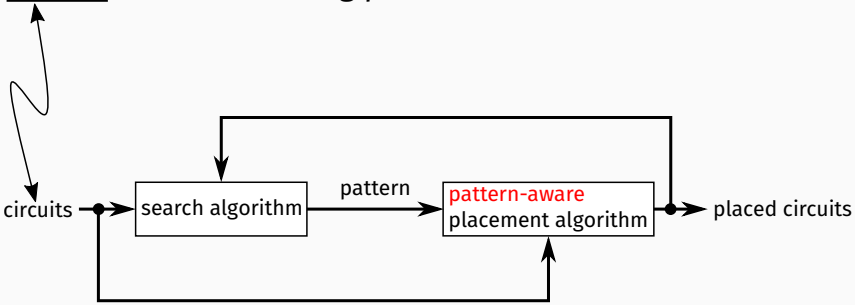
Exploration Philosophy

The Starting Premise

Circuits exhibit recurring patterns of interconnect

The Starting Premise

Circuits exhibit recurring patterns of interconnect



The Starting Premise

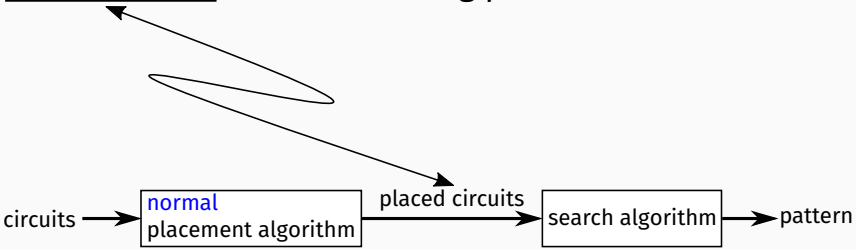
Circuits exhibit recurring patterns of interconnect

The Starting Premise

Placed circuits exhibit recurring patterns of interconnect

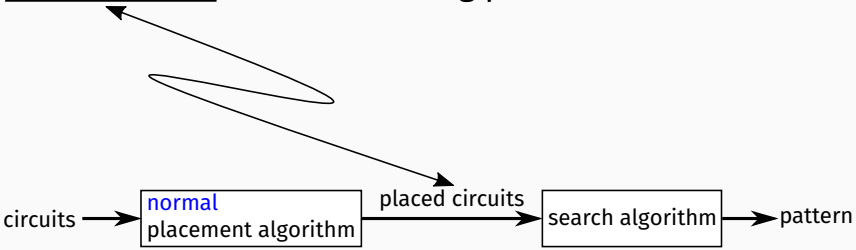
Opportunistic Direct Connection Usage

Placed circuits exhibit recurring patterns of interconnect



Opportunistic Direct Connection Usage

Placed circuits exhibit recurring patterns of interconnect



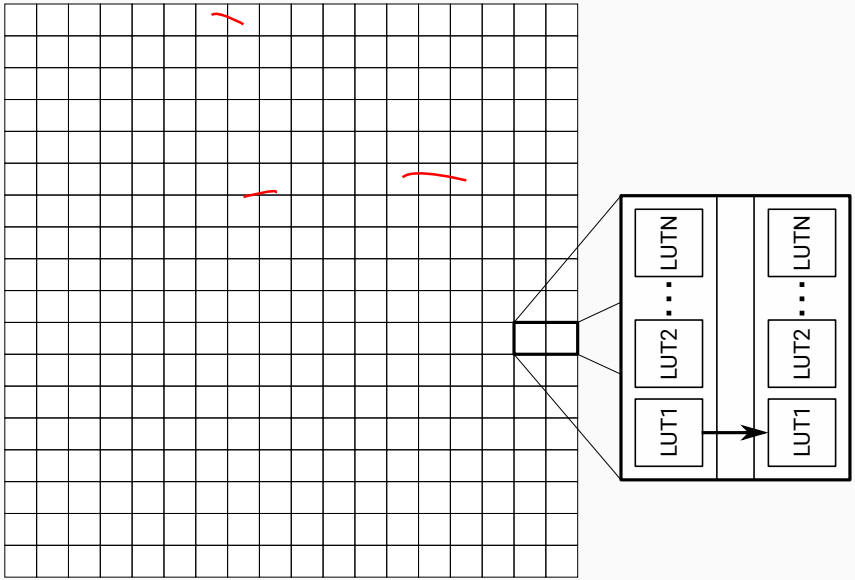
Pros:

- No need for new CAD
- No placement in the loop

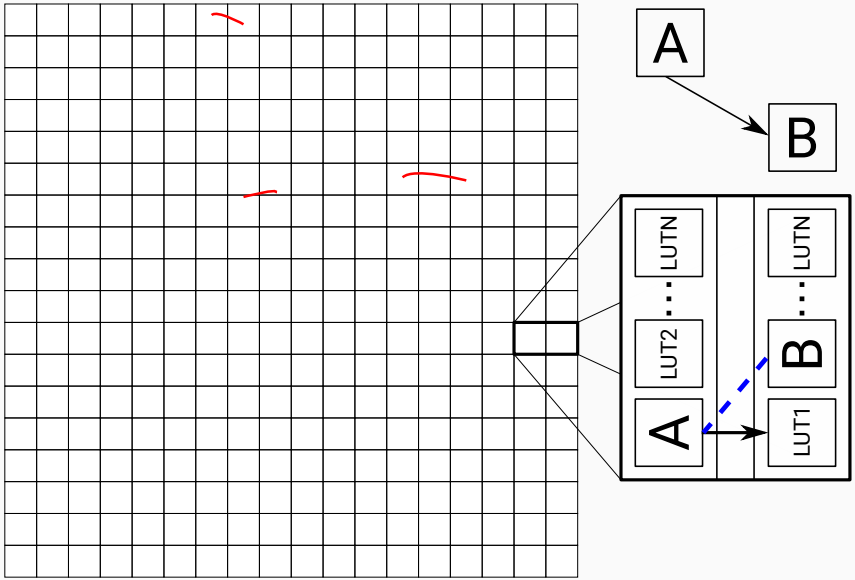
Cons:

- Some opportunities certainly missed

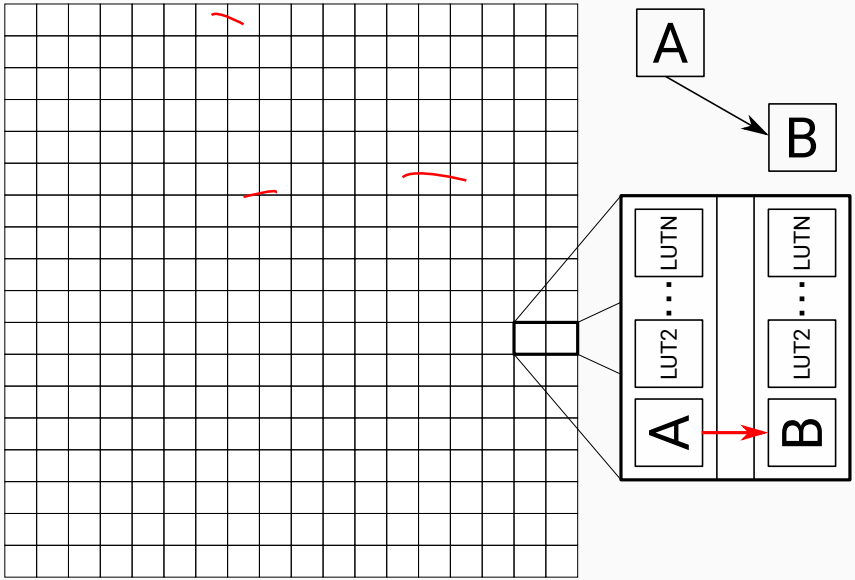
Opportunistic Direct Connection Usage: A Real Example (*sha*)



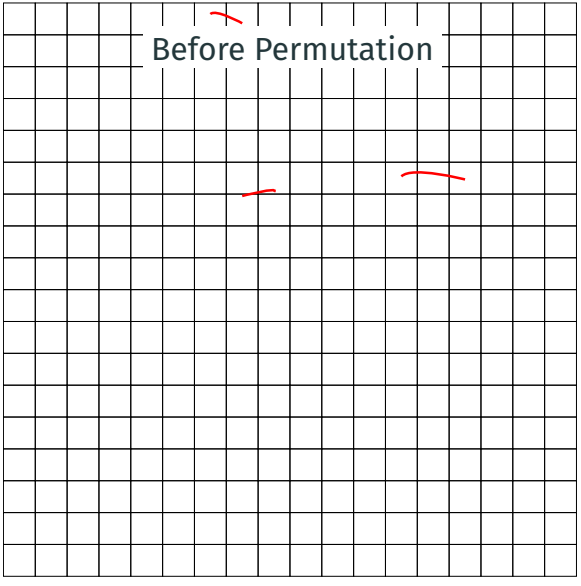
Opportunistic Direct Connection Usage: A Real Example (*sha*)

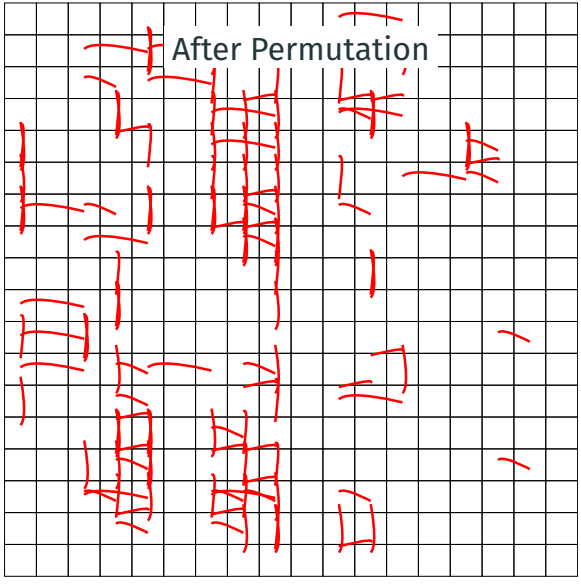


Opportunistic Direct Connection Usage: A Real Example (*sha*)

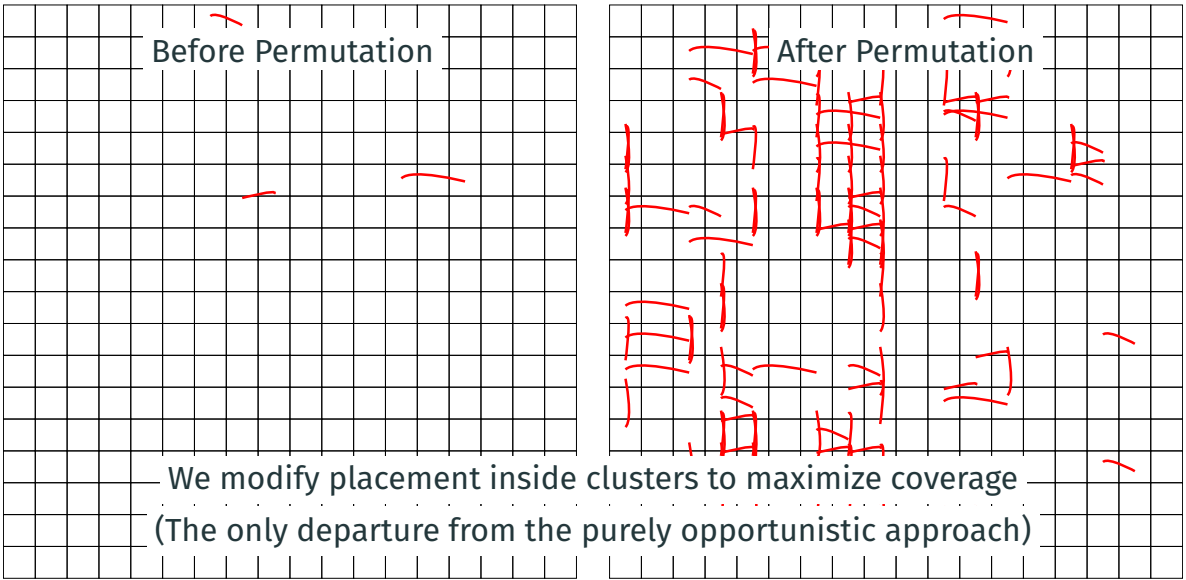


Opportunistic Direct Connection Usage: A Real Example (*sha*)





Opportunistic Direct Connection Usage: A Real Example (*sha*)



The Search Algorithm

General approach = enumerate + test

Enumeration: Some Constraints

- Pattern is the same for each tile
- (Chebyshev) length of the longest connection bounded by a constant w

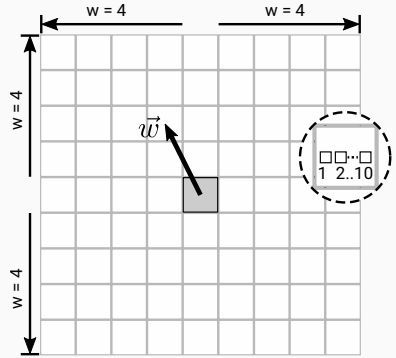
Enumeration: Problem Size

- 10 LUT cluster
- 20 direct connections
- $w = 4$

Enumeration: Problem Size

- 10 LUT cluster
- 20 direct connections
- $W = 4$

$$\#\text{edges} = \overbrace{10}^{\text{source LUT}} \times \overbrace{10}^{\text{target LUT}} \times \overbrace{81}^{\text{target cluster}}$$
$$= 8{,}100$$

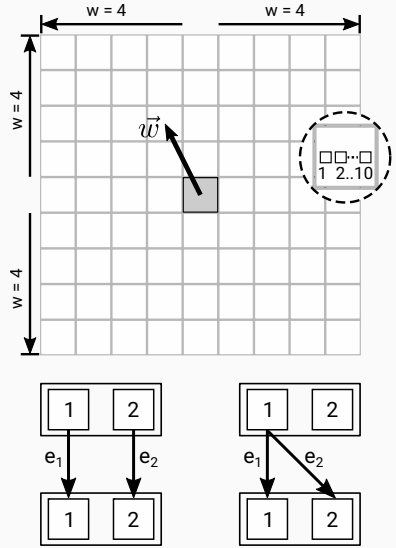


Enumeration: Problem Size

- 10 LUT cluster
- 20 direct connections
- $w = 4$

$$\#\text{edges} = \overbrace{10}^{\text{source LUT}} \times \overbrace{10}^{\text{target LUT}} \times \overbrace{81}^{\text{target cluster}}$$
$$= 8{,}100$$

$$\#\text{patterns} = \binom{8{,}100}{20} \sim 10^{59}$$



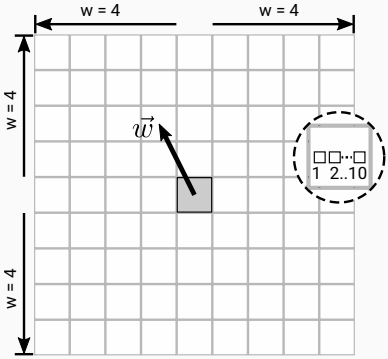
Enumeration: Problem Size

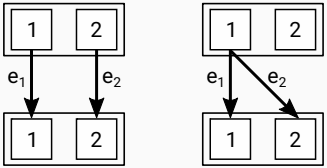
- 10 LUT cluster
- 20 direct connections
- $W = 4$

$$\#\text{edges} = \overbrace{10}^{\text{source LUT}} \times \overbrace{10}^{\text{target LUT}} \times \overbrace{81}^{\text{target cluster}} = 8{,}100$$

$$\#\text{patterns} = \binom{8{,}100}{20} \sim 10^{59}$$

Cannot be exhaustive...





A Greedy Approach

Some intuition behind the choice of approach in the paper

A Greedy Approach

Some intuition behind the choice of approach in the paper

1. List all additions of a single new direct connection to the current best pattern
2. Pick the best addition for the next iteration

A Greedy Approach

Some intuition behind the choice of approach in the paper

1. List all additions of a single new direct connection to the current best pattern
2. Pick the best addition for the next iteration

Best pattern = one with the lowest geomean delay

A Greedy Approach

Some intuition behind the choice of approach in the paper

1. List all additions of a single new direct connection to the current best pattern
2. Pick the best addition for the next iteration

Best pattern = one with the lowest geomean delay

$\Longrightarrow$ Still prohibitive for testing

(8,100 additions at each iteration $\Longrightarrow$ 162,000 architectures in total)

A Greedy Approach

Some intuition behind the choice of approach in the paper

1. List all additions of a single new direct connection to the current best pattern
2. Pick the best addition for the next iteration

Best pattern = one with the lowest geomean delay

$\Longrightarrow$ Still prohibitive for testing

(8,100 additions at each iteration $\Longrightarrow$ 162,000 architectures in total)

Apply filters to remove weak candidates

A Greedy Approach: Filtering

We apply three filters

A Greedy Approach: Filtering

We apply three filters

First two designed for speed and try to predict direct connection utilization, neglecting delay

A Greedy Approach: Filtering

We apply three filters

First two designed for speed and try to predict
direct connection utilization, neglecting delay

The third filter permutes LUTs inside their clusters
and updates the postplacement delay prediction accordingly

A Greedy Approach: Filtering

We apply three filters

First two designed for speed and try to predict
direct connection utilization, neglecting delay

The third filter permutes LUTs inside their clusters
and updates the postplacement delay prediction accordingly

Details about Filters 1 & 2 in the paper

The Third Filter (LUT Permutation)

Maximizing direct connection utilization is hard [1]

[1] T. Werth et al., “DAG Mining for Code Compaction”, Springer, 2009

The Third Filter (LUT Permutation)

Maximizing direct connection utilization is hard [1]

Postplacement critical path delay reduction often requires improving just a small fraction of connection delays

[1] T. Werth et al., “DAG Mining for Code Compaction”, Springer, 2009

The Third Filter (LUT Permutation)

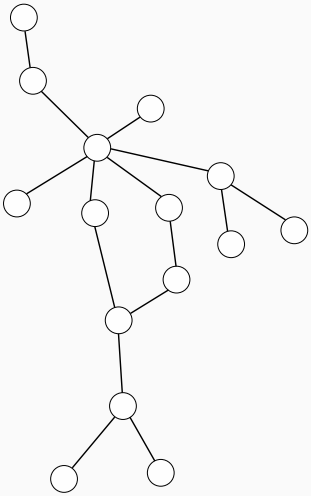
Maximizing direct connection utilization is hard [1]

Postplacement critical path delay reduction often requires improving just a small fraction of connection delays

$\Longrightarrow$ extract that fraction and form an ILP
(extract & solve the *critical core*)

[1] T. Werth et al., “DAG Mining for Code Compaction”, Springer, 2009

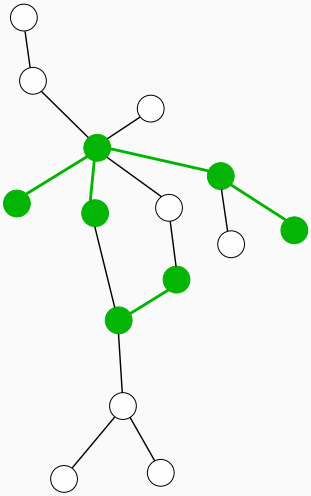
The Third Filter: Core Extraction



Timing graph

1. Core = all edges with a direct connection between endpoint clusters

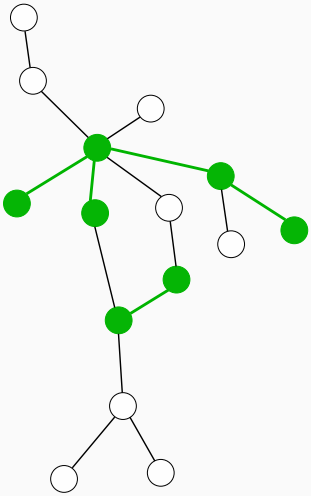
The Third Filter: Core Extraction



Timing graph

1. Core = all edges with a direct connection between endpoint clusters

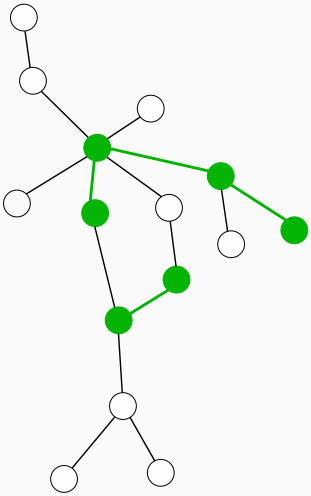
The Third Filter: Core Extraction



Timing graph

1. Core = all edges with a direct connection between endpoint clusters
2. Remove the edge of largest slack and least centrality

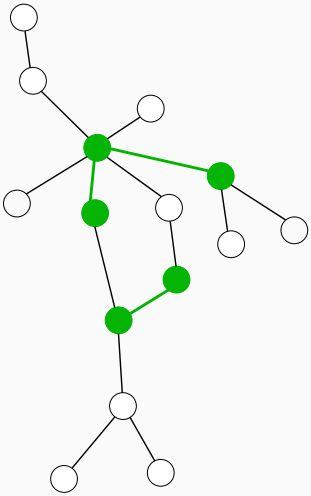
The Third Filter: Core Extraction



Timing graph

1. Core = all edges with a direct connection between endpoint clusters
2. Remove the edge of largest slack and least centrality

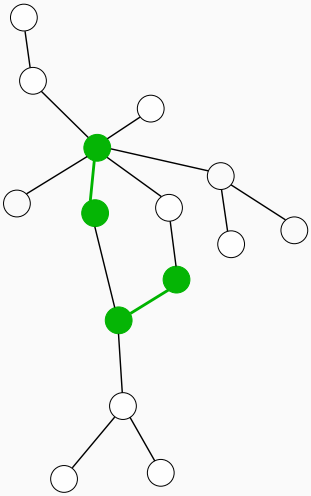
The Third Filter: Core Extraction



Timing graph

1. Core = all edges with a direct connection between endpoint clusters
2. Remove the edge of largest slack and least centrality

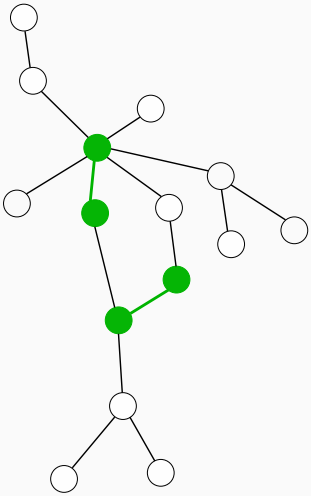
The Third Filter: Core Extraction



Timing graph

1. Core = all edges with a direct connection between endpoint clusters
2. Remove the edge of largest slack and least centrality

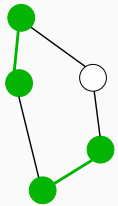
The Third Filter: Core Extraction



Timing graph

1. Core = all edges with a direct connection between endpoint clusters
2. Remove the edge of largest slack and least centrality
3. Crop to nodes on paths between the core-nodes

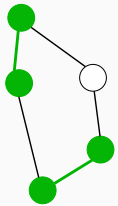
The Third Filter: Core Extraction



Timing graph

1. Core = all edges with a direct connection between endpoint clusters
2. Remove the edge of largest slack and least centrality
3. Crop to nodes on paths between the core-nodes

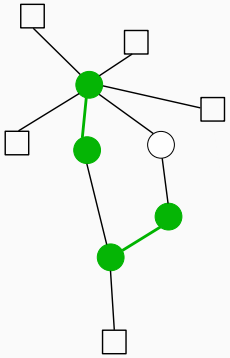
The Third Filter: Core Extraction



Timing graph

1. Core = all edges with a direct connection between endpoint clusters
2. Remove the edge of largest slack and least centrality
3. Crop to nodes on paths between the core-nodes
4. Constrain the periphery

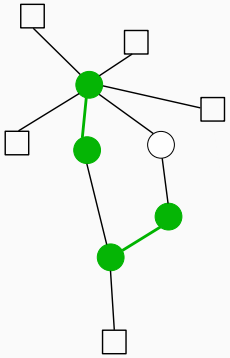
The Third Filter: Core Extraction



Timing graph

1. Core = all edges with a direct connection between endpoint clusters
2. Remove the edge of largest slack and least centrality
3. Crop to nodes on paths between the core-nodes
4. Constrain the periphery

The Third Filter: Core Solving (ILP)

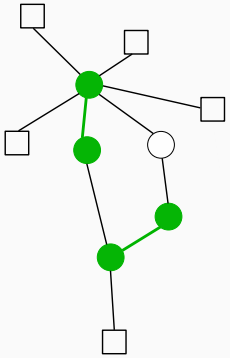


Timing graph

LUT positions:

$$\forall u \in \text{Core}, p \in [0, N] : \quad x_{u,p} \in \{0, 1\}$$

The Third Filter: Core Solving (ILP)



Timing graph

LUT positions:

$$\forall u \in \text{Core}, p \in [0, N] : \quad x_{u,p} \in \{0, 1\}$$

Edge delays:

$$\forall (u, v) \in \text{Core}, p_1, p_2 \in [0, N] :$$
$$t_{d_{u,v}} = \sum t_{up_1,vp_2} x_{u,p_1} x_{v,p_2}$$

Experimental Setup

Experimental Setup

k6_N10_mem32K_40nm VTR 7.0 architecture used as underlying

A subset of VTR benchmarks is used

All results medians of 5 placement seeds

Everything routed with *delay-targeted routing algorithm* [1]

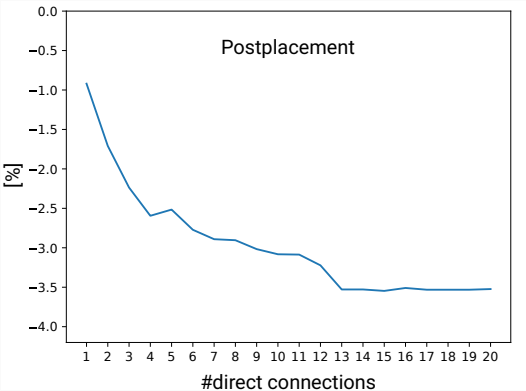
[1] R. Rubin, A. DeHon, “Timing-Driven Pathfinder Pathology and Remediation: Quantifying and Reducing Delay Noise in VPR-Pathfider”, FPGA’11

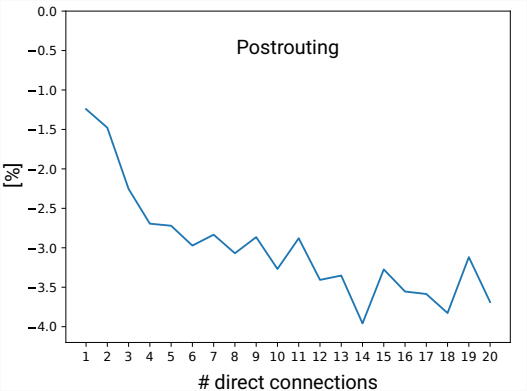
Limitations

No support for carry chains, fracturable LUTs, and sparse crossbars (multipliers and memories supported)

Results

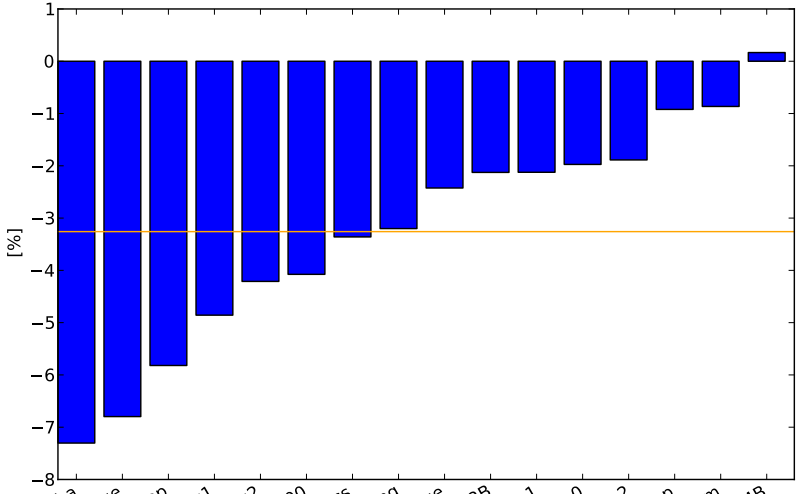
Convergence



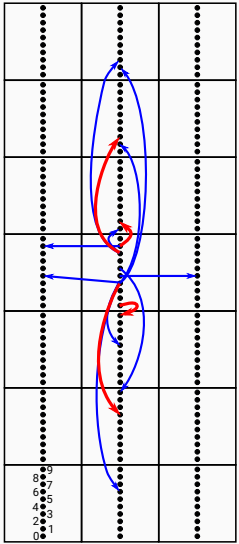


Evolution of geomean delay change with addition of direct connections

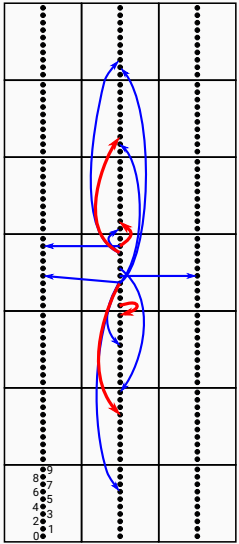
Delay Impact



The Pattern

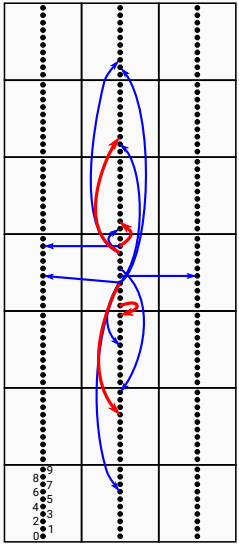


The Pattern



~ 1% cluster area increase

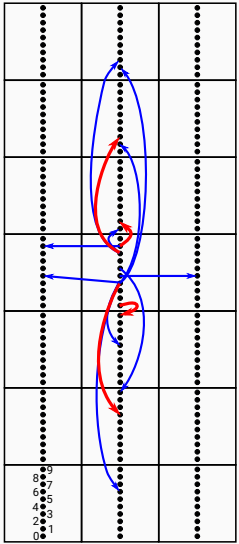
The Pattern



~ 1% cluster area increase

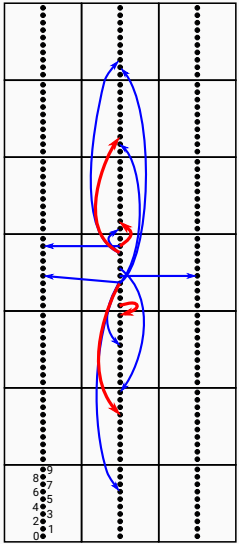
Broadcasting all 14 connections to all 60 crossbar muxes (cluster-cluster case) would cost a lot more

The Pattern



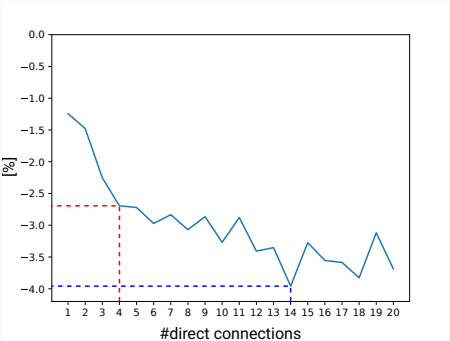
Red edges = first four added

The Pattern

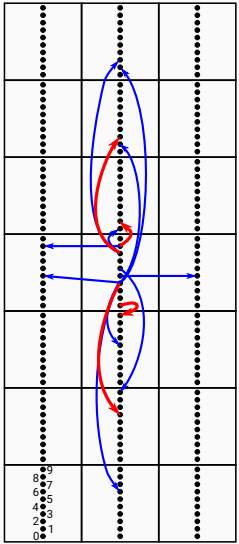


Red edges = first four added

68% achieved delay improvement for $< 0.3\%$ cluster area increase



The Pattern

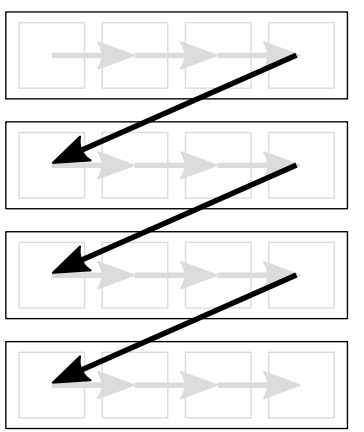


Red edges = first four added

68% achieved delay improvement for
$< 0.3\%$ cluster area increase

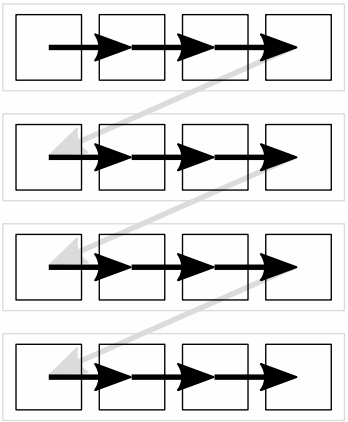
Any usage forms a matching in the circuit
$\Longrightarrow$ possibly easy mapping

Two-Stage Search



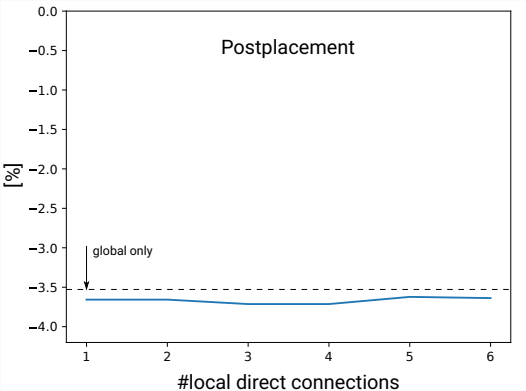
First stage: intercluster (global) connections

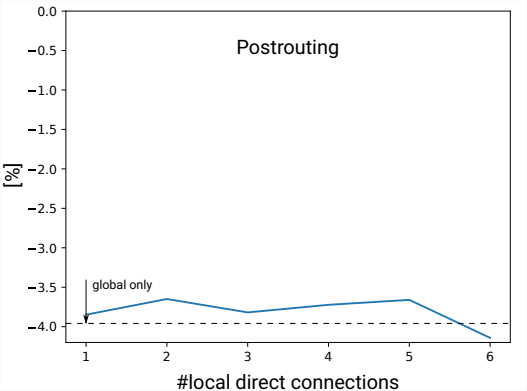
Two-Stage Search



Second stage: intracluster (local) connections

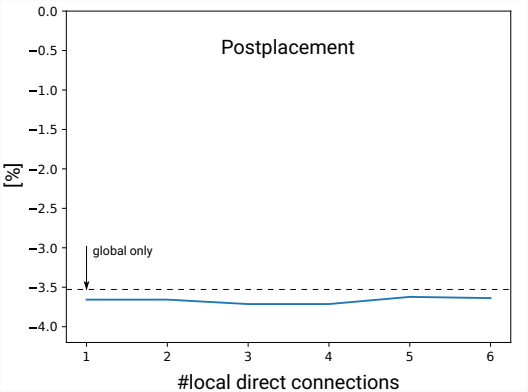
Convergence: Intracluster

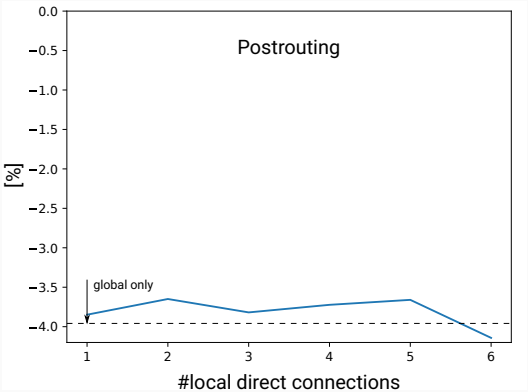




Local connections added on top of existing global ones

Convergence: Intracluster





Not that appealing...

Conclusions and Future Work

Conclusions

Complex wire hardening pays off!

Developed an efficient algorithm that finds good patterns to harden

Future Work

How much further could we go if we had dedicated CAD tools?

Thank you for attention