Regularity Matters: Designing Practical FPGA Switch-Blocks

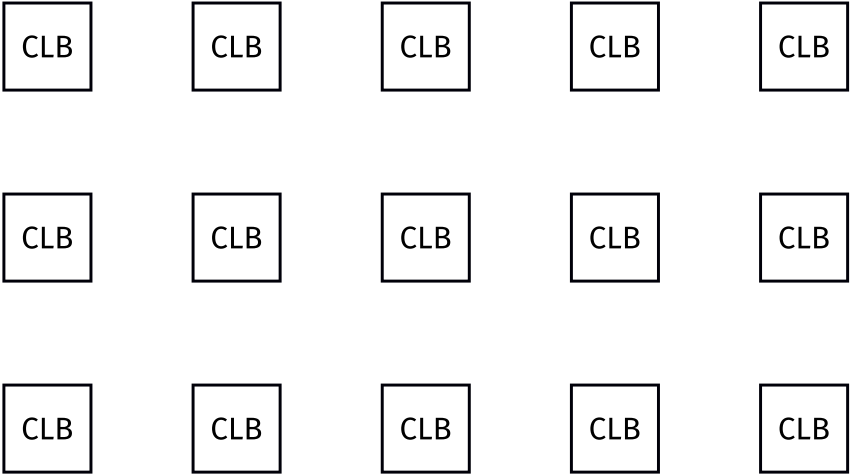
Stefan Nikolić and Paolo Ienne

FPGA'23, Monterey, 13.02.2023

École Polytechnique Fédérale de Lausanne

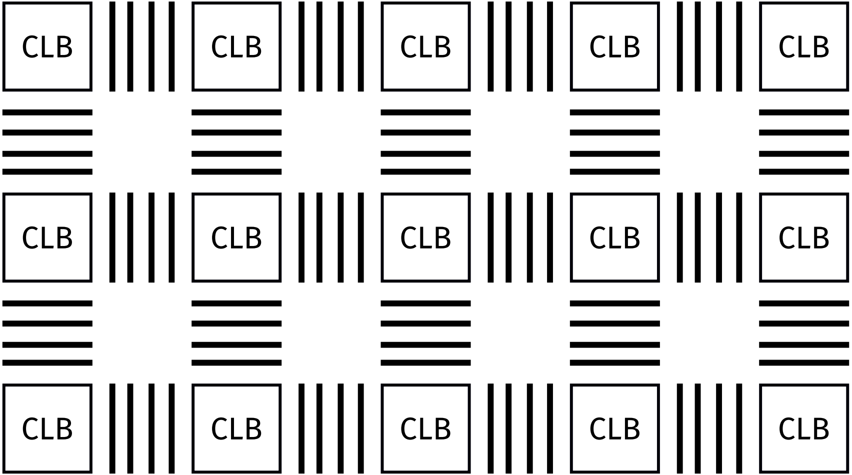
What is the problem?

A very quick review of Island-Style FPGA architecture



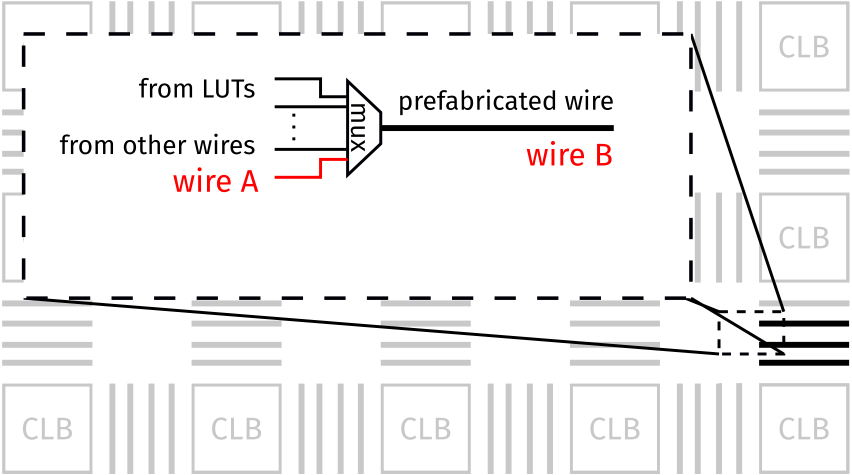
"Islands of LUTs"

A very quick review of Island-Style FPGA architecture



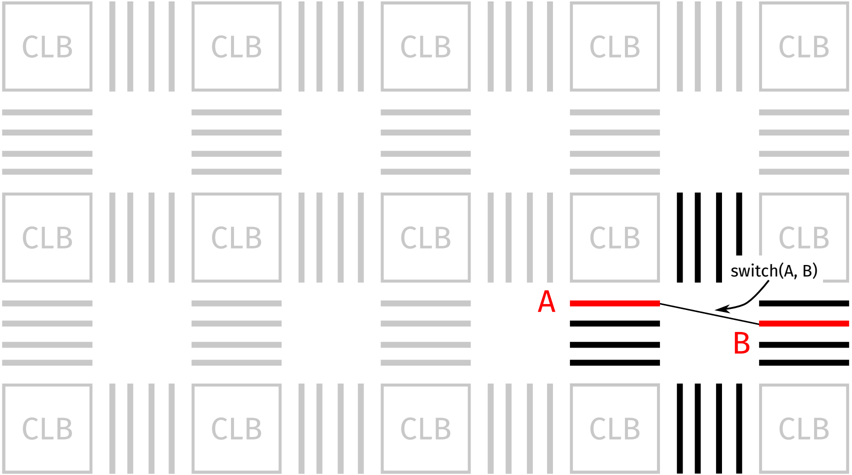
“Surrounded by channels of prefabricated wires”

A very quick review of Island-Style FPGA architecture



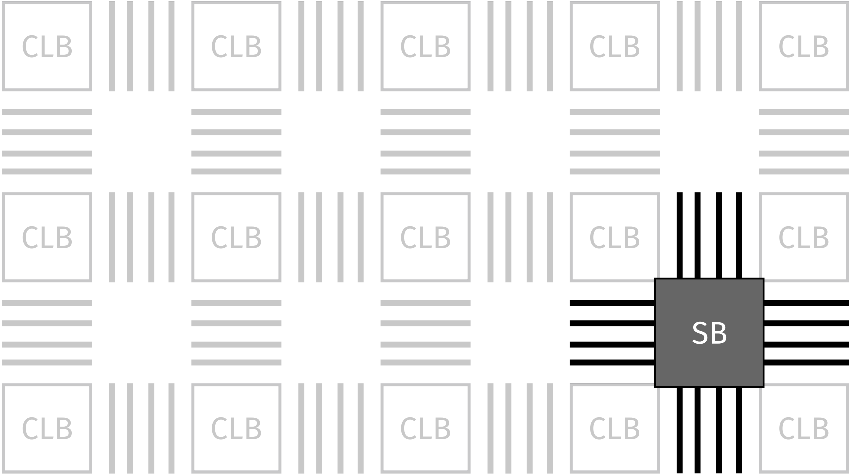
“There is a **SWITCH** between wire A and wire B”

A very quick review of Island-Style FPGA architecture



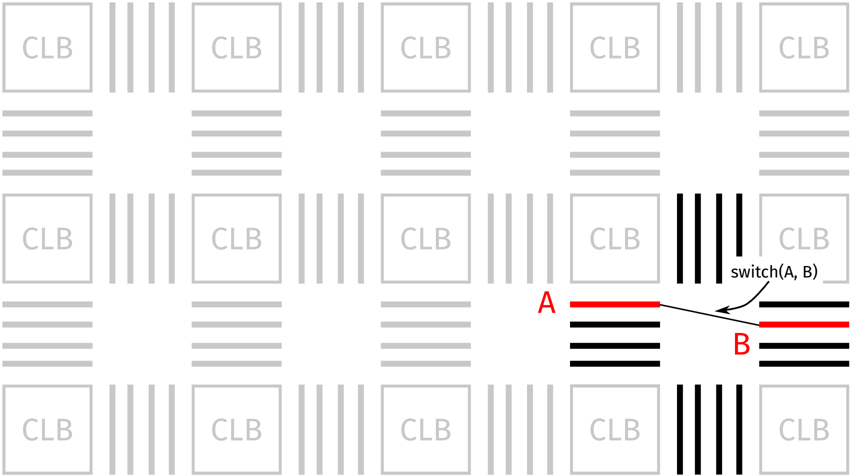
“There is a **SWITCH** between wire A and wire B”

A very quick review of Island-Style FPGA architecture



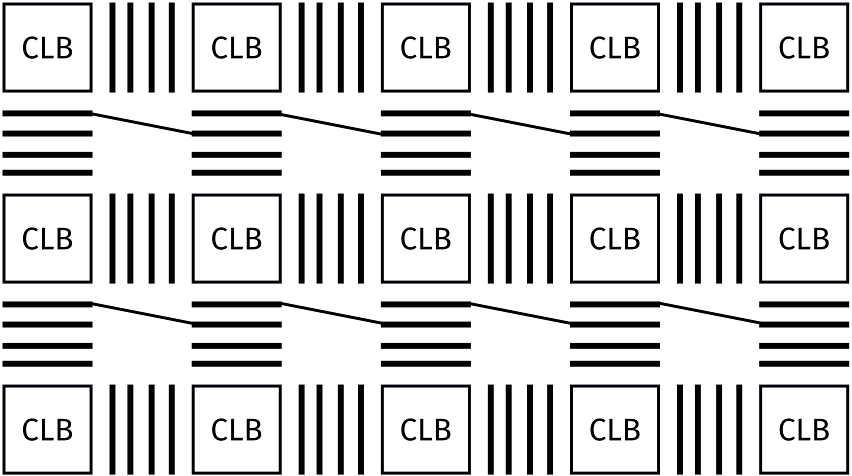
SWITCH-BLOCK (SB)

A very quick review of Island-Style FPGA architecture



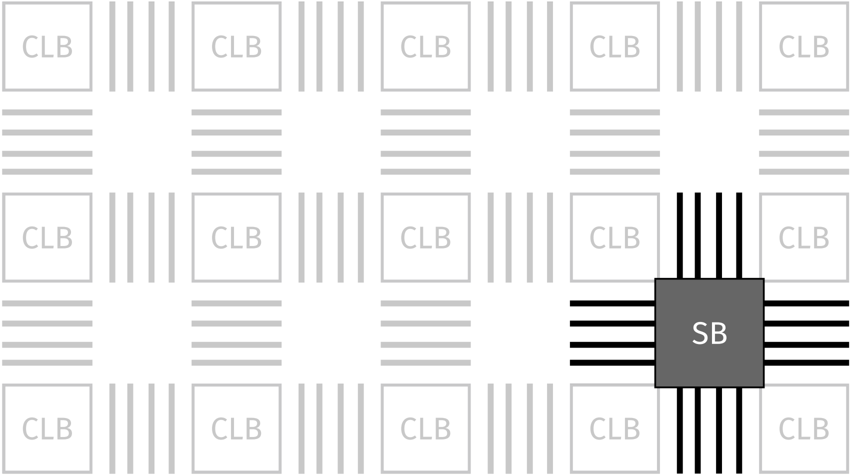
All SBs are identical (e.g., one has switch(A, B) $\implies$ all have them

A very quick review of Island-Style FPGA architecture



All SBs are identical (e.g., one has switch(A, B) $\implies$ all have them

What is the problem?

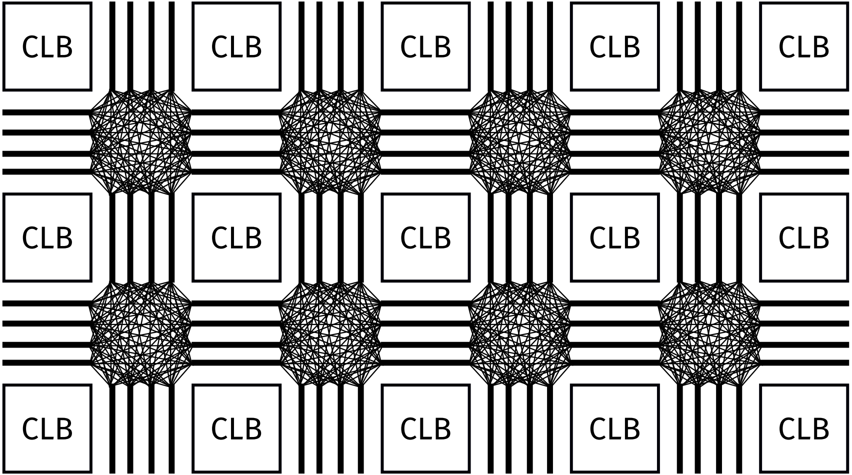


Which switches should the SB have?

What do we expect from a switch-block?

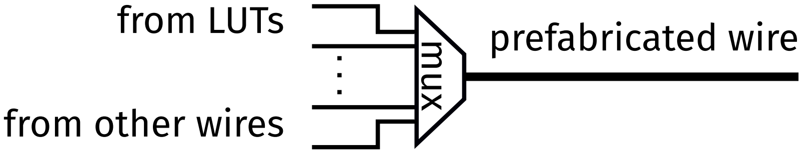
Route many different connections
of many different circuits
with minimal delay

What do we expect from a switch-block?

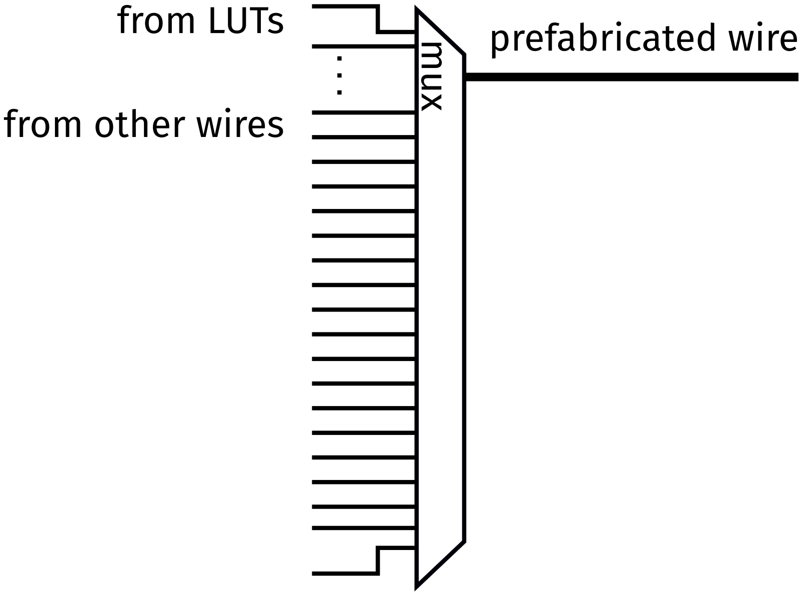


fully-connected $\Longrightarrow$ hops always minimal

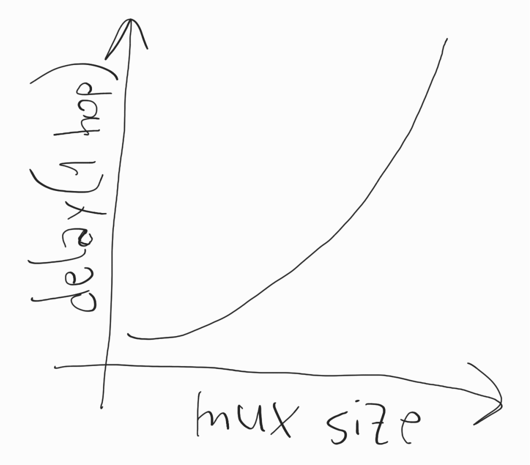
What do we expect from a switch-block?



What do we expect from a switch-block?



What do we expect from a switch-block?

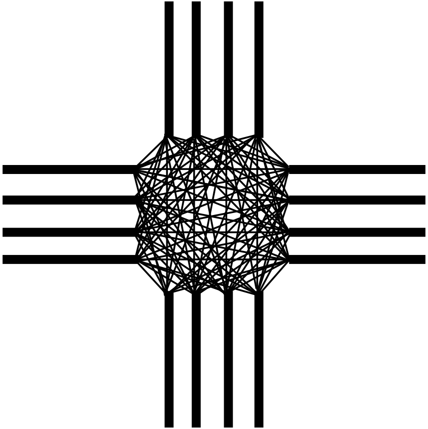


What do we expect from a switch-block?

So we need

- something sparse
- that still minimizes hops
- allows paths to avoid intersecting (congestion resolution)

What is the problem?



How to make this sparse? (and REGULAR)

Isn’t switch-pattern design a closed problem?

2019 International Conference on Field-Programmable Technology (ICFPT)

A Study on Switch Block Patterns for Tileable FPGA Routing Architectures

Xifan Tang, Edouard Giacomin, Aurélien Alacchi and Pierre-Emmanuel Gaillardon
University of Utah
Email: xifan.tang@utah.edu

***Abstract*—Following the rapid growth of *Field Programmable Gate Arrays* (FPGAs) sizes, the regularity of architectures has become a critical feature, leading to the development of million-of-LUT devices. While the routing architecture plays a dominant role in the area, delay and power of modern FPGAs, most of previously published works focus on improving the routability and performance of FPGAs while very few studied tileable (highly-regular) routing architectures. In this paper, we provide a detailed analysis between tileable and popular non-tileable FPGAs considering modern routing architectures. First, we upgrade VPR to generate tileable routing architecture, which can support different switch block patterns for (1) the routing tracks that start/end in a tile and (2) the routing tracks that pass through a tile. Then, we evaluate the performance of mixed switch blocks patterns in the context of a Stratix IV-like FPGA architecture, by considering the most representative patterns, i.e., Subset, Universal and Wilton. Experimental results show that averaged over the MCNC and VTR benchmarks, when compared to the well-optimized non-tileable architectures, the tileable architectures can improve the minimum routable channel width by 13% and area-delay product by 2%. In particular, our results showed that in the context of tileable FPGA, a mix of Universal and Wilton switch block patterns lead to the best trade-**

Experimental results show that compared to VPR, our RRG generator can reduce the number of unique tiles by $8.8\times$ and $5.5\times$ for homogeneous and heterogeneous FPGAs respectively, even considering 128×128 array size.

(2) More than tileable FPGA, our RRG generator also supports different switch block patterns for (a) the routing tracks that start/end in a tile and (b) the routing tracks that pass a tile. We evaluate the performance of mixed switch blocks patterns in the context of a Stratix IV-like FPGA architecture, by considering the most representative patterns, i.e., Subset [20], Universal [19], Wilton [21] and Imran [22]. Experimental results show that averaged over the MCNC and VTR benchmarks, when compared to the well-optimized non-tileable architectures, the tileable architectures can improve the minimum routable channel width by 13% and area-delay product by 2%. In particular, our results showed that in the context of tileable FPGA, a mix of Universal and Wilton switch block patterns leads to the best trade-off in area, delay and routability, while Wilton switch block was the best choice in non-tileable FPGAs.

Isn’t switch-pattern design a closed problem?

2019 International Conference on Field-Programmable Technology (ICFPT)

A Study on Switch Block Patterns for Tileable FPGA Routing Architectures

Xifan Tang, Edouard Giacomin, Aurélien Alacchi and Pierre-Emmanuel Gaillardon
University of Utah
Email: xifan.tang@utah.edu

***Abstract*—Following the rapid growth of *Field Programmable Gate Arrays* (FPGAs) sizes, the regularity of architectures has become a critical feature, leading to the development of million-of-LUT devices. While the routing architecture plays a dominant role in the area, delay and power of modern FPGAs, most of previously published works focus on improving the routability and performance of FPGAs while very few studied tileable (highly-regular) routing architectures. In this paper, we provide a detailed analysis between tileable and popular non-tileable FPGAs considering modern routing architectures. First, we upgrade VPR to generate tileable routing architecture, which can support different switch block patterns for (1) the routing tracks that start/end in a tile and (2) the routing tracks that pass through a tile. Then, we evaluate the performance of mixed switch blocks patterns in the context of a Stratix IV-like FPGA architecture, by considering the most representative patterns, i.e., Subset, Universal and Wilton. Experin that averaged over the MCNC and VTR compared to the well-optimized non-tileabl tileable architectures can improve the minimum routable channel width by 13% and area-delay product by 2%. In particular, our results showed that in the context of tileable FPGA, a mix of Universal and Wilton switch block patterns lead to the best trade-**

Experimental results show that compared to VPR, our RRG generator can reduce the number of unique tiles by 8.8× and 5.5× for homogeneous and heterogeneous FPGAs respectively, even considering 128 × 128 array size.
(2) More than tileable FPGA, our RRG generator a different switch block patterns for (a) the routing start/end in a tile and (b) the routing tracks that pass a tile. We evaluate the performance of mixed switch blocks patterns in the context of a Stratix IV-like FPGA architecture, by considering the most representative patterns, i.e., Subset [20] Universal [19], Wilton [21] and Imran [22]. Experimental results show that averaged over the MCNC and VTR benchmarks, when compared to the well-optimized non-tileable architectures, ... a mix of Un ... Wilton switch b... ns ... to the best trade ... a, delay and rout ... ile Wilton switch block was the best choice in non-tileable FPGAs.

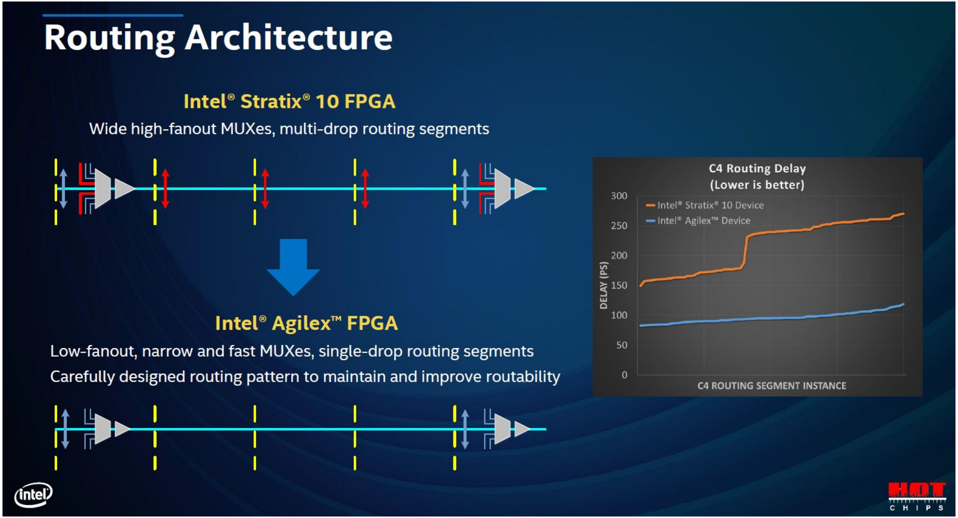
Subset [20] 1997

Universal [19] 1996

Wilton [21] 1997

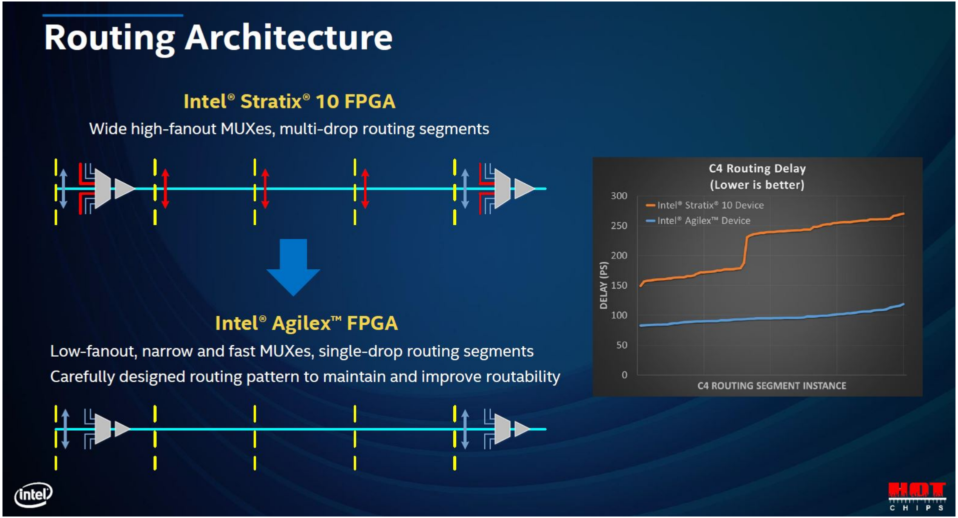
Imran [22] 1999

Meanwhile in industry: technology driving change



[1] Ganusov and Iyer. Agilex Generation of Intel FPGAs. Hot Chips'20

Meanwhile in industry: technology driving change



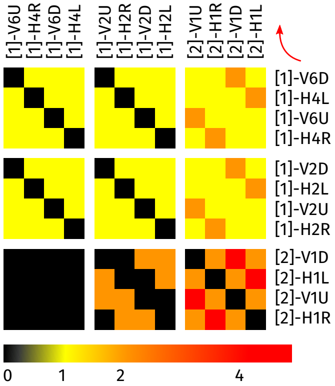
"Carefully designed routing pattern to maintain and improve routability"

Can we automatically design switch-patterns optimized for technology X?

[1] Lin, Wawrzynek, El Gamal. Exploring FPGA routing architecture stochastically. TCAD'10
[2] Nikolić and Ienne. Turning PathFinder Upside-Down. FPL'21. **Best Paper Award**

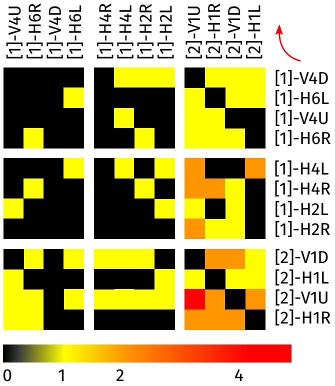
Irregularity problem

Regular



7-Series [1]

Irregular



Avalanche search [2]

[1] Petersen, Nikolić, and Stojilović. NetCracker: A Peek into... 7-Series FPGAs. FPGA'21

[2] Nikolić and Ienne. Turning PathFinder Upside-Down. FPL'21. **Best Paper Award**

Two questions

Assuming that regularity is necessary for layout reasons

1. How do we ensure that the algorithm always finds a regular solution?
2. How does the solution quality change when regularity is enforced?

Presentation outline

Motivation

A review of Avalanche Search

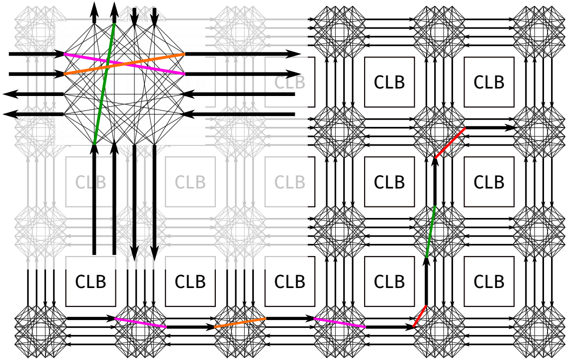
Enforcing regularity

Costs and benefits of regularity

A review of Avalanche Search?

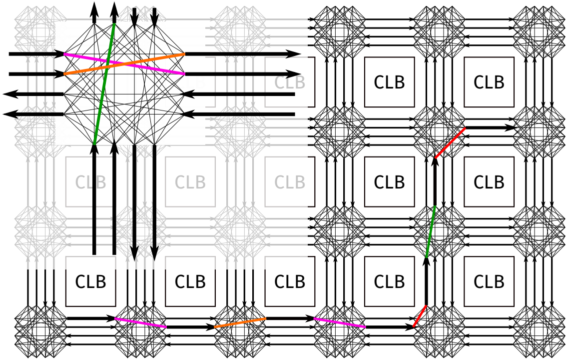
FPGA router vs mosaic artist

Disclaimer: authors have nothing to do with Damanhur; photo had a creative commons license



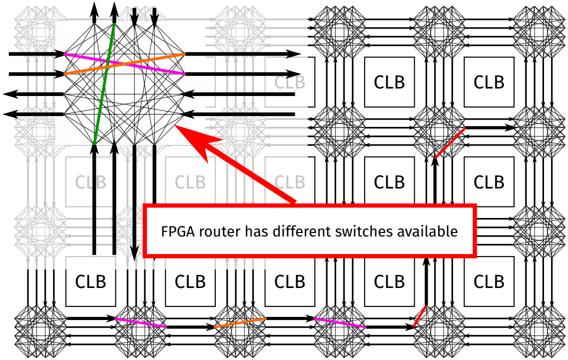
FPGA router vs mosaic artist

Disclaimer: authors have nothing to do with Damanhur; photo had a creative commons license

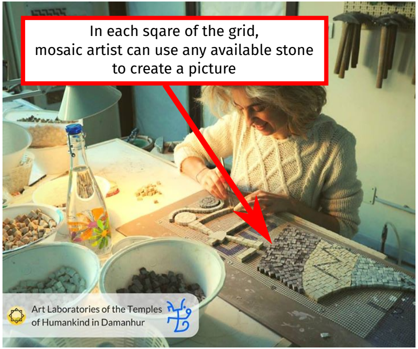


FPGA router vs mosaic artist

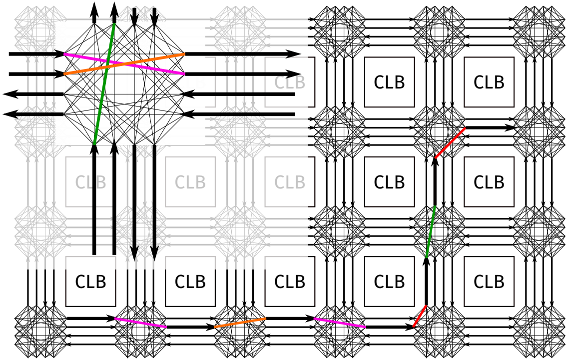
Disclaimer: authors have nothing to do with Damanhur; photo had a creative commons license



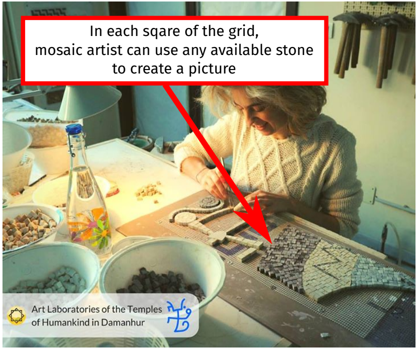
FPGA router vs mosaic artist



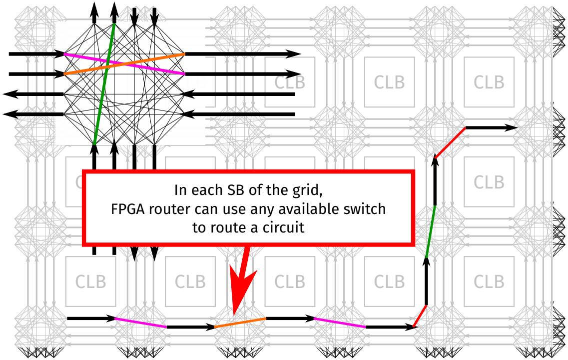
Disclaimer: authors have nothing to do with Damanhur; photo had a creative commons license



FPGA router vs mosaic artist



Disclaimer: authors have nothing to do with Damanhur; photo had a creative commons license



FPGA architect vs seller of mosaic stones

FPGA architect's problem

- Too many different switches = expensive, poor performance

FPGA architect vs seller of mosaic stones

FPGA architect's problem

- Too many different switches = expensive, poor performance

Mosaic stone seller's problem

- Too many different stones = hard to search through inventory

FPGA architect vs seller of mosaic stones

FPGA architect's problem

- Too many different switches =
  expensive, poor performance

FPGA architect's goal

- Minimize the number
  of different switches
- While making the router happy
  (so that it creates fast circuits)

Mosaic stone seller's problem

- Too many different stones =
  hard to search through inventory

FPGA architect vs seller of mosaic stones

FPGA architect's problem

- Too many different switches = expensive, poor performance

FPGA architect's goal

- Minimize the number of different switches
- While making the router happy (so that it creates fast circuits)

Mosaic stone seller's problem

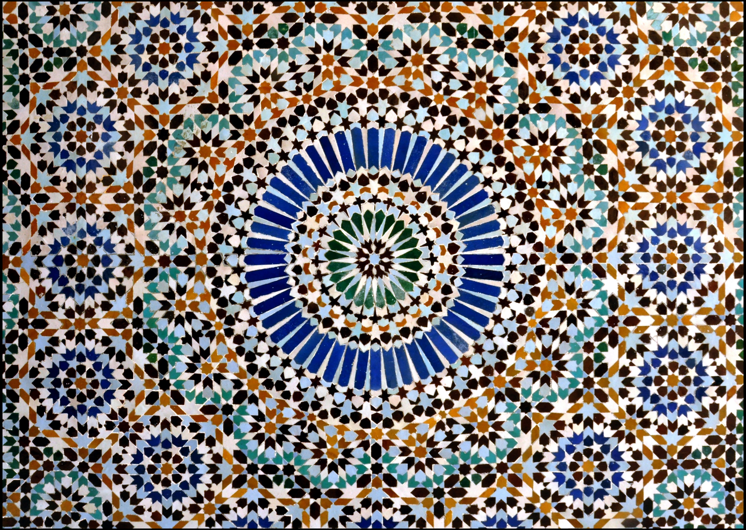
- Too many different stones = hard to search through inventory

Mosaic stone seller's goal

- Minimize the number of different stones
- While making the artist happy (so that they create nice mosaics)

Mosaic stone sellers existed centuries before FPGA architects…

Ahmed is a 16th century mosaic artist



Ahmed's ideal design made of 1000 different kinds of stones

Ahmed sails to Constantinople to buy the stones

He goes to a bazaar to find the store selling 1000 kinds of stones

… And presents his list to Mustafa the shop owner

1. Ultramarine medium vedge (576 pieces)
2. Alabaster needle (229 pieces)
3. Coral hexagon (143 pieces)

.

.

.

1000. Verdigris tiny star (2 pieces)

When Mustafa saw “1000. Verdigris tiny star (2 pieces)”

“Oh my, will I really have to search through all these bags?”

Mustafa has an idea

1. Ultramarine medium vedge (576 pieces)
2. Alabaster needle (229 pieces)
3. Coral hexagon (143 pieces)

.
.
.

10.

1000. Verdigris tiny star (2 pieces)

“Here, you seem to need these 10 kinds the most.”

Mustafa has an idea

1. Ultramarine medium vedge (576 pieces)
2. Alabaster needle (229 pieces)
3. Coral hexagon (143 pieces)

⋮

10\.

1000\. Verdigris tiny star (2 pieces)

"I give you a big discount on any number of them!"

Mustafa has an idea

1. Ultramarine medium vedge (576 pieces)
2. Alabaster needle (229 pieces)
3. Coral hexagon (143 pieces)

⋮

10.

1000. Verdigris tiny star (2 pieces)

"But once I take the bags out, you have to buy. Deal?"

Mustafa has an idea

1. Ultramarine medium vedge (576 pieces)
2. Alabaster needle (229 pieces)
3. Coral hexagon (143 pieces)

⋮

10.

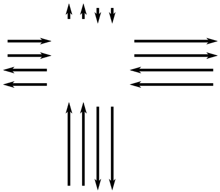
1000. Verdigris tiny star (2 pieces)

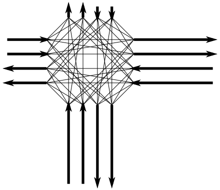
"You can come tomorrow with a new design."

Ahmed agrees and tries to redesign the mosaic to maximize the usage of discounted stones

Let’s see how this works for switch-block exploration

Initial setting

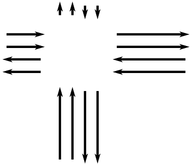




PathFinder routes the first time (Ahmed creates his ideal mosaic)

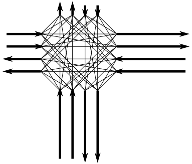
switches marked for fabrication
(kinds of stone Mustafa pledged to bring out)

discounted cost
(0)

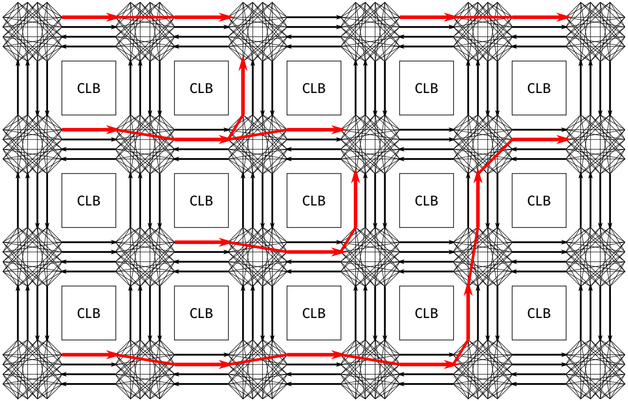


switches available to the router
(Mustafa's catalogue)

everything else
is expensive

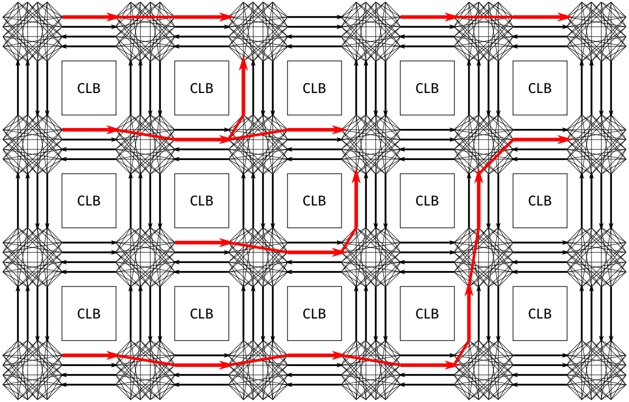


PathFinder routes circuits (each connection by a shortest path)
(Ahmed designs the mosaic)

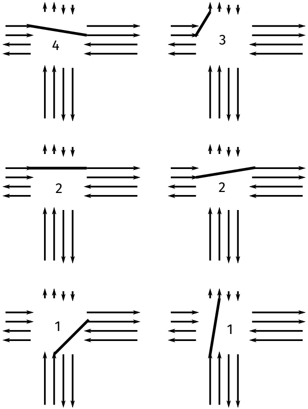


Count in how many SBs each switch is used (Ahmed writes his list)

PathFinder routes circuits (each connection by a shortest path)
(Ahmed designs the mosaic)

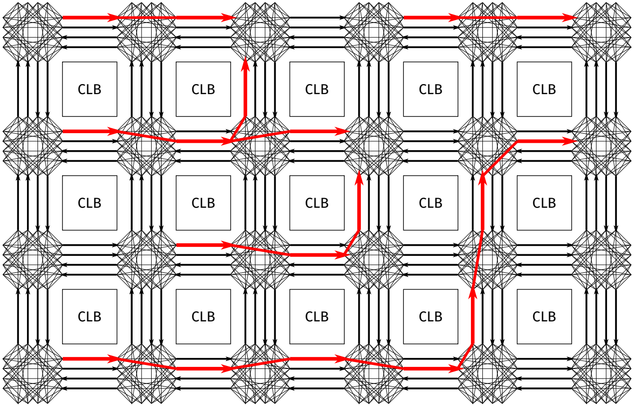


Count in how many switch-blocks each switch was used
(Ahmed counts different stones)

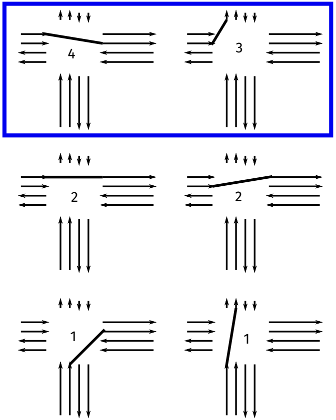


Give discounts and mark for fabrication (Mustafa gives discounts)

PathFinder routes circuits (each connection by a shortest path)
(Ahmed designs the mosaic)



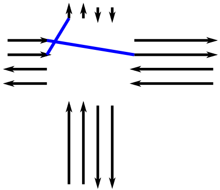
Give discounts and mark for fabrication
(Mustafa gives discount and takes out the bags)



Give discounts and mark for fabrication (Mustafa gives discounts)

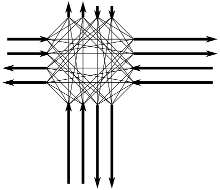
switches marked for fabrication
(kinds of stone Mustafa pledged to bring out)

discounted cost
(0)



switches available to the router
(Mustafa's catalogue)

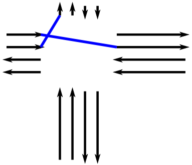
everything else
is expensive



PathFinder routes again, maximizing usage of discounted switches (Ahmed redesigns his mosaic)

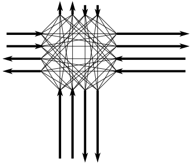
switches marked for fabrication
(kinds of stone Mustafa pledged to bring out)

discounted cost
(0)

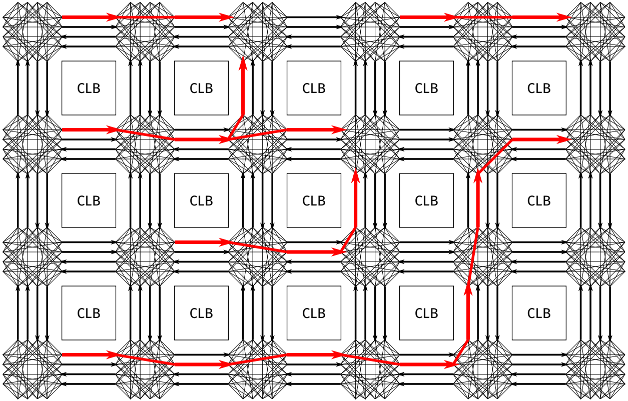


switches available to the router
(Mustafa's catalogue)

everything else
is expensive



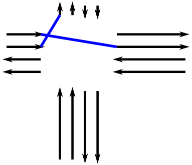
PathFinder routes circuits (each connection by a shortest path)
(Ahmed designs the mosaic)



PathFinder routes again, maximizing usage of discounted switches (Ahmed redesigns his mosaic)

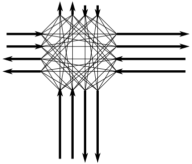
switches marked for fabrication
(kinds of stone Mustafa pledged to bring out)

discounted cost
(0)

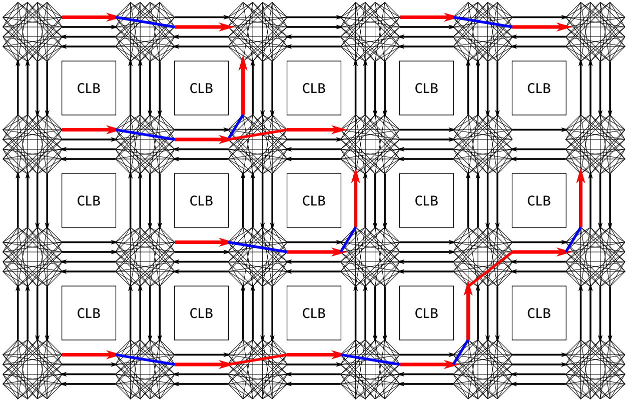


switches available to the router
(Mustafa's catalogue)

everything else
is expensive

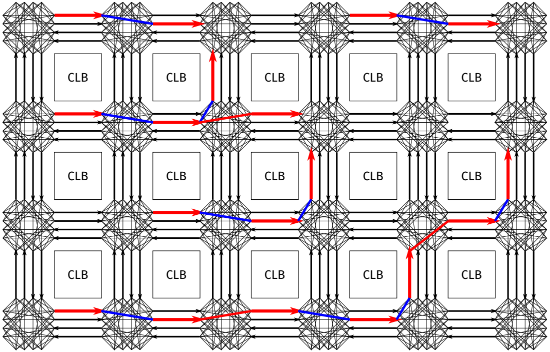


PathFinder routes circuits (each connection by a shortest path)
(Ahmed designs the mosaic)

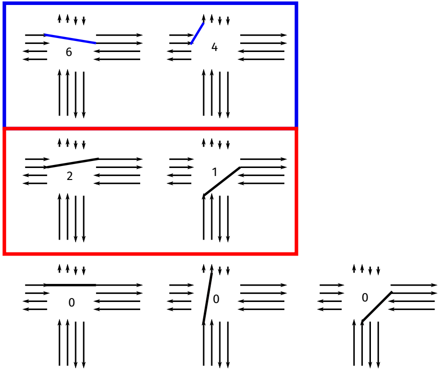


PathFinder still wants non-discounted switches (Ahmed's list is still long)

PathFinder routes circuits (each connection by a shortest path)
(Ahmed designs the mosaic)



Count in how many switch-blocks each switch was used
(Ahmed counts different stones)

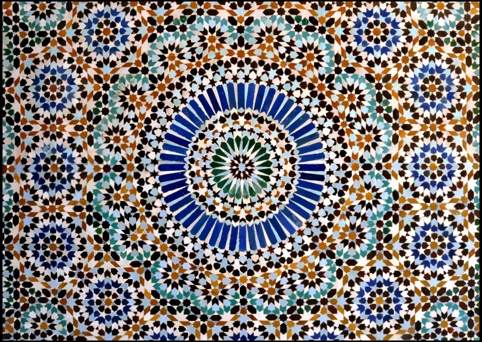


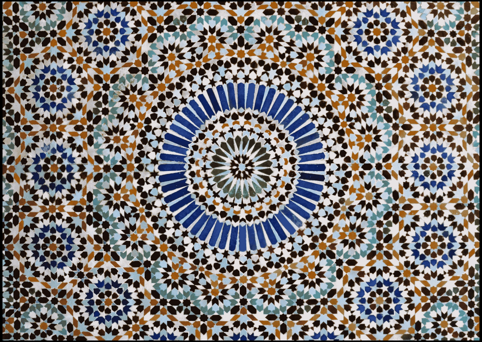
Could be required for connectivity, congestion, or critical paths

What do we do now?

Let’s see what Mustafa did

Ahmed agrees and tries to rework the design thinking of the discount





With only 493 kinds of stones, difference is barely visible

So Ahmed goes to bargain with Mustafa again

1. Ultramarine medium vedge
2. Alabaster needle
3. Coral hexagon

⋮

10.

493. Sienna medium triangle (6 pieces)

When Mustafa saw “493. Sienna medium triangle (6 pieces)”

“Oh my, here we go again...”

But he still wants Ahmed to be happy

1. Ultramarine medium vedge
2. Alabaster needle
3. Coral hexagon

.

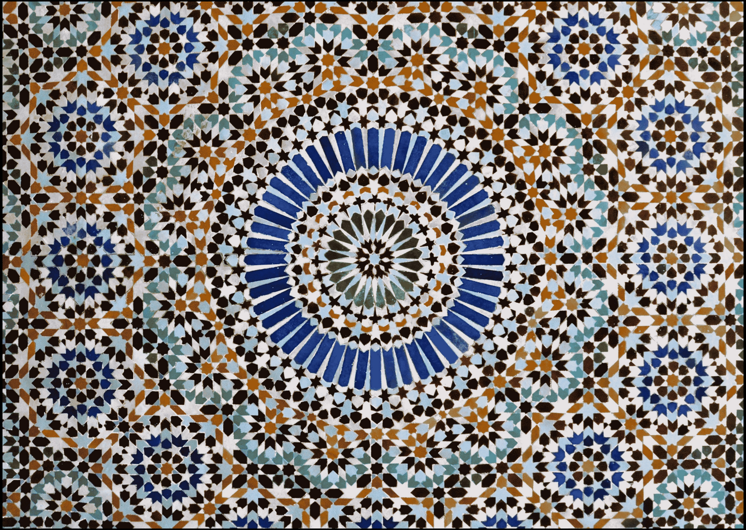
10. .

20. .

493. Sienna medium triangle (6 pieces)

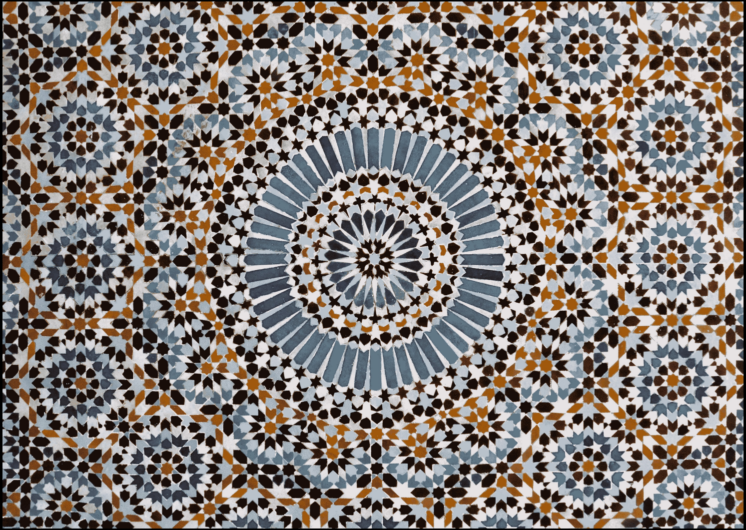
"Alright, alright, I give you these next ten with a large discount too."

Ahmed goes to rework the mosaic further



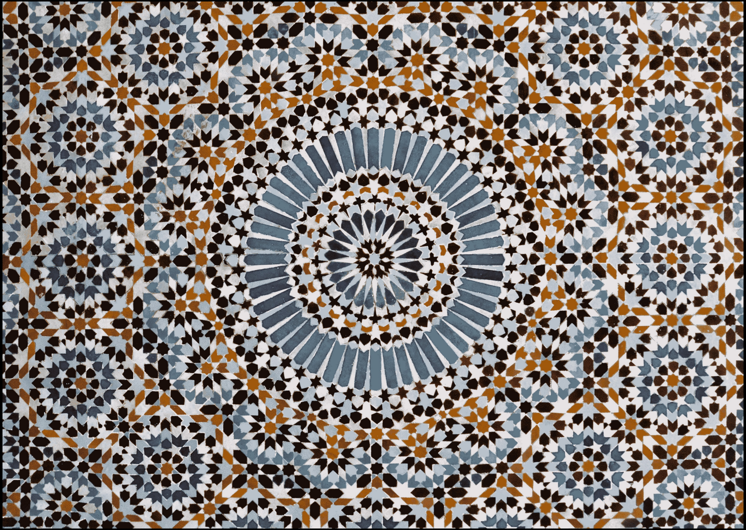
Then he bargains with Mustafa some more

An agreement is made



Eventually, Ahmed is happy with his new design taking advantage of Mustafa's discounts

An agreement is made

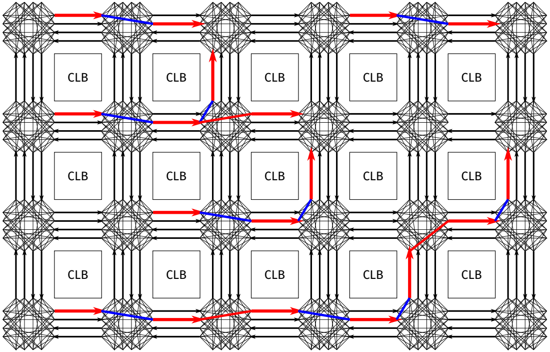


By then, Mustafa took out 87 bags—quite some work, but $\ll 1000$

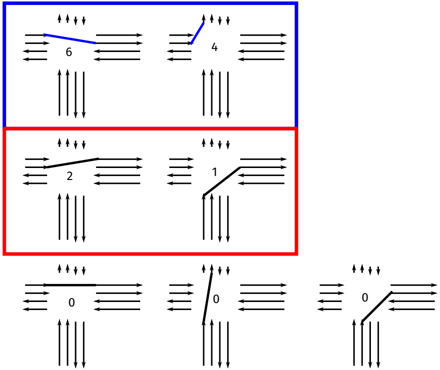
What do we do now?

Give new discounts

PathFinder routes circuits (each connection by a shortest path)
(Ahmed designs the mosaic)

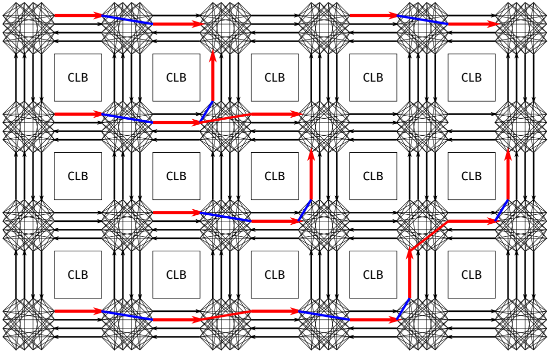


Count in how many switch-blocks each switch was used
(Ahmed counts different stones)

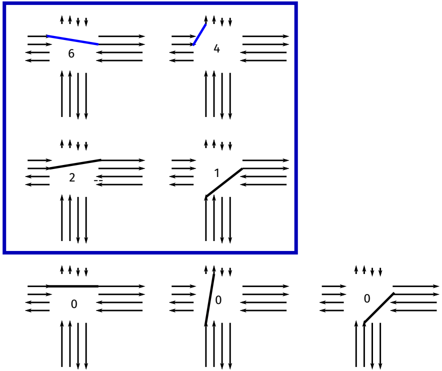


Give new discounts

PathFinder routes circuits (each connection by a shortest path)
(Ahmed designs the mosaic)



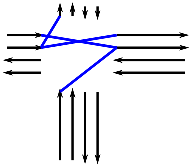
Give discounts and mark for fabrication
(Mustafa gives discount and takes out the bags)



PathFinder uses only switches marked for fabrication (blue)

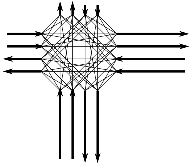
switches marked for fabrication
(kinds of stone Mustafa pledged to bring out)

discounted cost
(0)

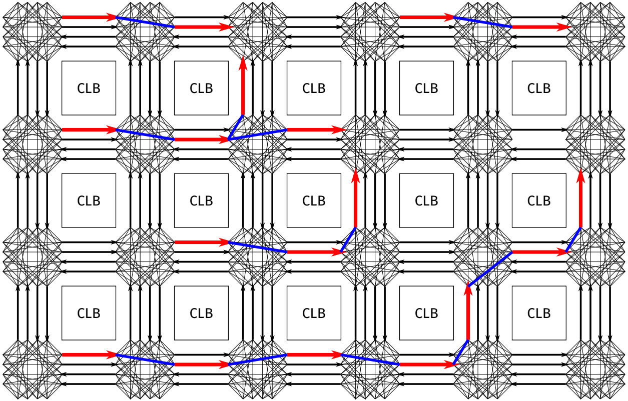


switches available to the router
(Mustafa's catalogue)

everything else
is expensive

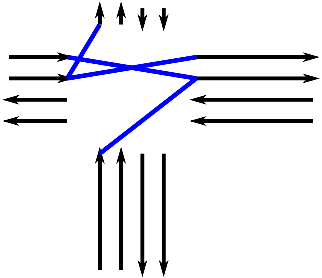


PathFinder routes circuits (each connection by a shortest path)
(Ahmed designs the mosaic)



Everybody is happy

And we have the final pattern that will be produced



Avalanche Search [Nikolić and Ienne. Turning PathFinder Upside-Down. FPL'21]

1. switches marked for fabrication $\leftarrow \{\}$
2. all possible switches can be used at cost C
3. let PathFinder route the circuits
   (with additional switch-minimization costing; see FPL'21)
4. if no unmarked switches are used, done
5. mark n most-used unmarked switches and set their cost to 0
6. goto 3

And this is how we obtain IRREGULAR patterns...

Even if we manage to minimize the switch block to M switches

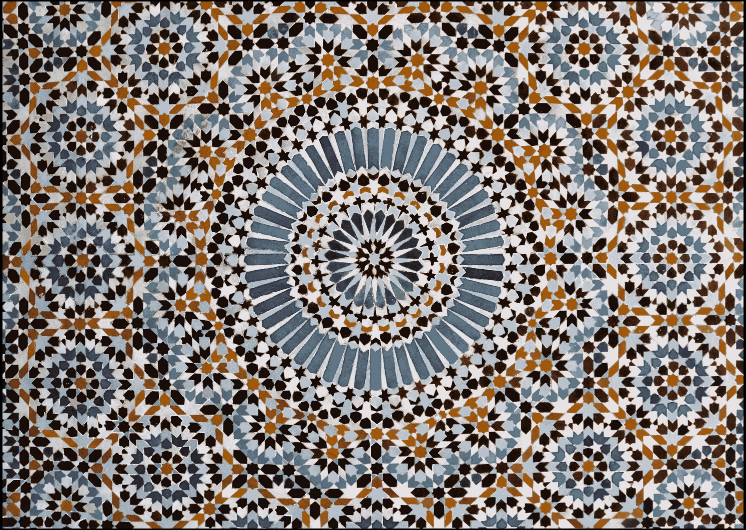
1. some M switches will be easy to lay out
2. others will be hard (or impossible) to lay out

How do we find M switches that are easy to lay out, but as close as possible to the M that the router wants?

Enforcing regularity

Mustafa had a similar problem

An agreement is made



By then, Mustafa took out 87 bags, $\ll 1000$

An agreement is made

But, to take out these 87, he had to move 492 bags in total

An agreement is made

Since the ones he pledged to take out were scattered around the pile

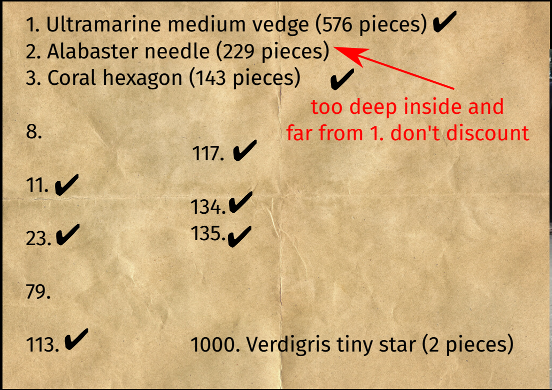
Mustafa's new problem

"How do I give Ahmed 87 kinds of stone that are as close as possible to the ones he wants, but so that I never move more than 150 bags?"

Two years after, Ahmed sails to Constantinople again

But Mustafa is smarter this time

Location of bags matters


1. Ultramarine medium vedge (576 pieces) ✔
2. Alabaster needle (229 pieces)
3. Coral hexagon (143 pieces) ✔

8.

11. ✔

23. ✔

79.

113. ✔

117. ✔

134. ✔

135. ✔

1000. Verdigris tiny star (2 pieces)

Mustafa ticks 87 bags that Ahmed desires the most,
but which don't violate his constraints (on moving $\leqslant 150$ bags)

Location of bags matters

1. Ultramarine medium vedge (576 pieces) ✔
2. Alabaster needle (229 pieces)
3. Coral hexagon (143 pieces) ✔

too deep inside and far from 1. don't discount

8.

11. ✔

23. ✔

79.

113. ✔

117. ✔

134. ✔

135. ✔

1000. Verdigris tiny star (2 pieces)

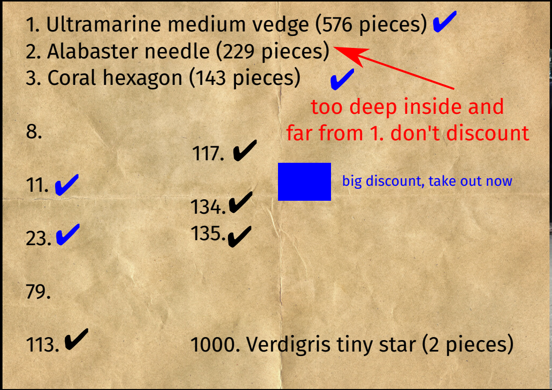
big discount, take out now

“For stone kinds ticked in blue, I give you a big discount.
For any number of pieces!”

Location of bags matters

“But once I take the bags out, you have to buy. Deal?”

Constraints are hard



Mustafa always ends bargaining on a complete ticked list

Important takeaways

Exact constraints which Mustafa applies while ticking Ahmed’s list depend on the layout of his pile

Important takeaways

For another merchant they will be different

Important takeaways

But the algorithm is identical
(and hence general)

Let's see how this works for **REGULAR** switch-block exploration

Key points: constructing a feasible solution

1. Encode **ANY** regularity constraints
   (e.g., for layout or CAD tools)
   as an Integer Linear Program (ILP)

2. Let ILP maximize PathFinder’s “desire” (usage of different switches)
   while satisfying the above constraints

Key points: updating costs

For switches in the ILP solution (satisfying regularity constraints)

1. Give a big discount to *n* most-used unmarked switches

For switches not in the ILP solution (violating regularity constraints)

1. Assign full cost with no discount

Key points: ensuring that constraints are met

- Final pattern is the last ILP solution

Regularizing Avalanche Search

1. switches marked for fabrication $\leftarrow \{\}$
2. all possible switches can be used at cost C
3. let PathFinder route the circuits
4. if iterations expanded $\implies$ return the last ILP solution
5. solve the ILP, always retaining marked switches
6. mark n most-used unmarked switches from ILP's solution and set their cost to 0
7. goto 3

Costs and benefits of regularity

Experimental setup

- A plane-based architecture with eight 6-LUTs in the CLB
- $2\times$ H1, H2, H4, H6, and $2\times$ V1, V4 wires per LUT
- 4nm predictive technology [1]
- alu4, tseng, ex5p MCNC [2] circuits ($\sim$ 2 700 LUTs in total) simultaneously routed in exploration by VTR-8
- MCNC circuits for critical path delay measurement
- 10 000 LUT Gnl [3] circuits for routability measurement

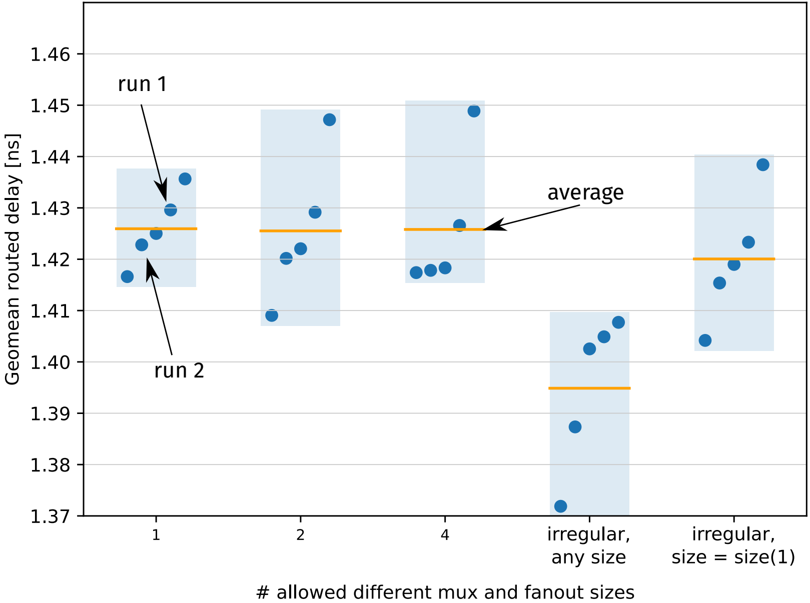
[1] Nikolić, Catthoor, Tőkei, and Ienne. Global Is the New Local. FPGA'21
[2] Yang, Logic synthesis and optimization benchmarks user guide. MCNC Tech. Report'91
[3] Stroobandt, Depreitere, and Campenhout. Generating new benchmark designs. Integration'99

Limiting the number of different multiplexer and fanout sizes

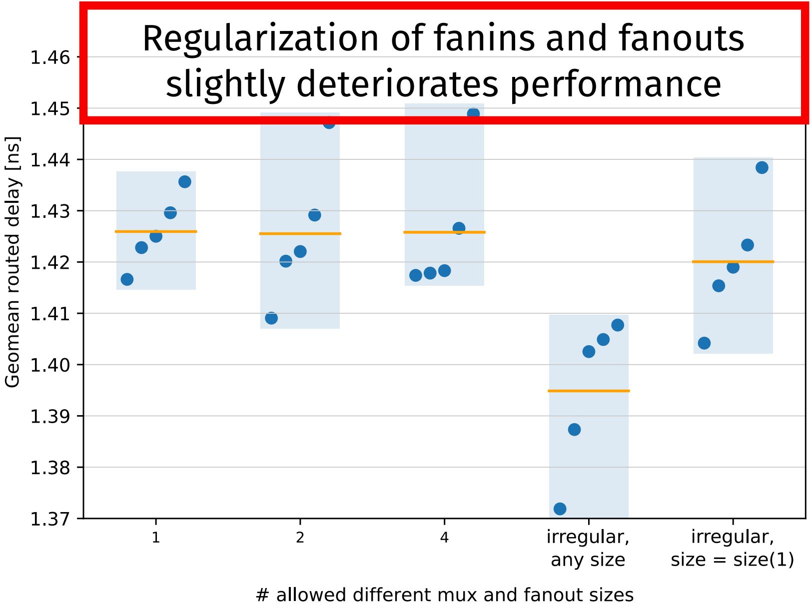
- Fewer multiplexer sizes $\implies$ easier layout
- Observed in commercial architectures [1]

[1] Petersen, Nikolić, and Stojilović. NetCracker: A Peek into… 7-Series FPGAs. FPGA'21

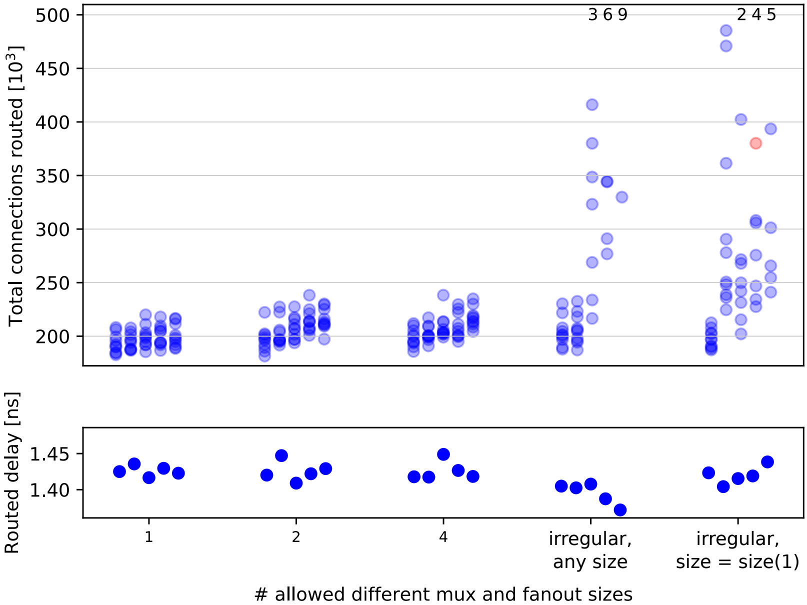
Multiplexer and fanout sizes: delay



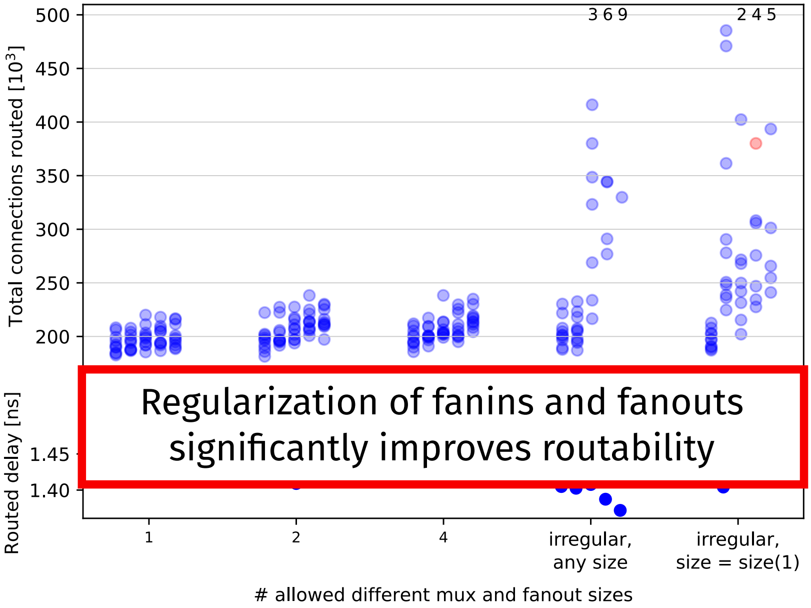
Multiplexer and fanout sizes: delay



Multiplexer and fanout sizes: routability



Multiplexer and fanout sizes: routability



Regularization of fanins and fanouts significantly improves routability

Input sharing

Each mux m_1 shares $\geqslant \xi$ inputs with at least one other mux m_2

(reduced capacitance, vias, area)

[1] Chromczak, Wheeler, Chiasson, How, Langhammer, Vanderhoek, Zgheib, and Ganusov. Architectural Enhancements in Intel® Agilex™ FPGAs. FPGA'20

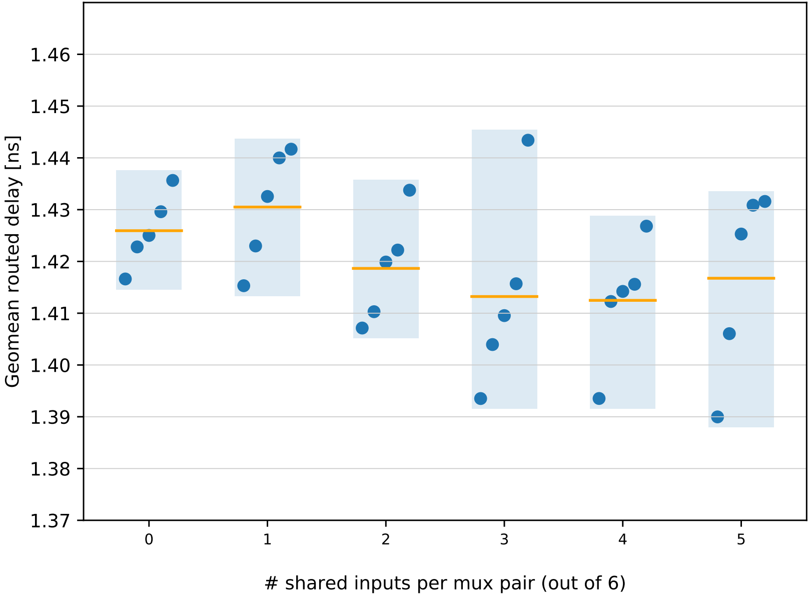
[2] Petersen, Nikolić, and Stojilović. NetCracker: A Peek into... 7-Series FPGAs. FPGA'21

Input sharing: experiments

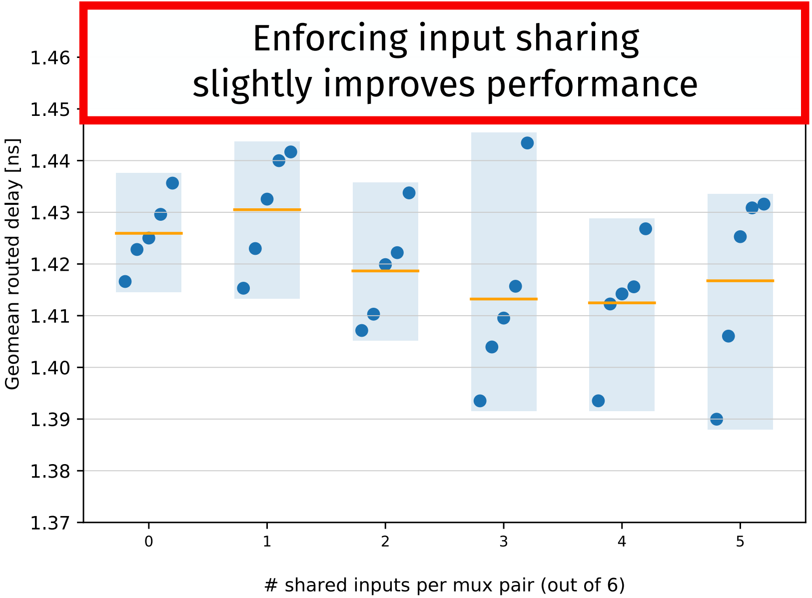
All wires have a fanin and fanout of 6 to other wires

sharing $\in [0..5]$

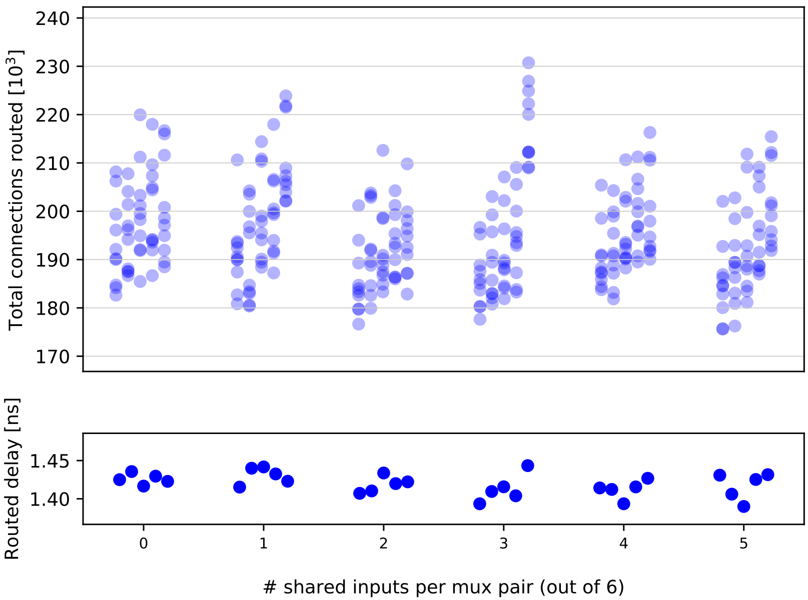
Input sharing: delay



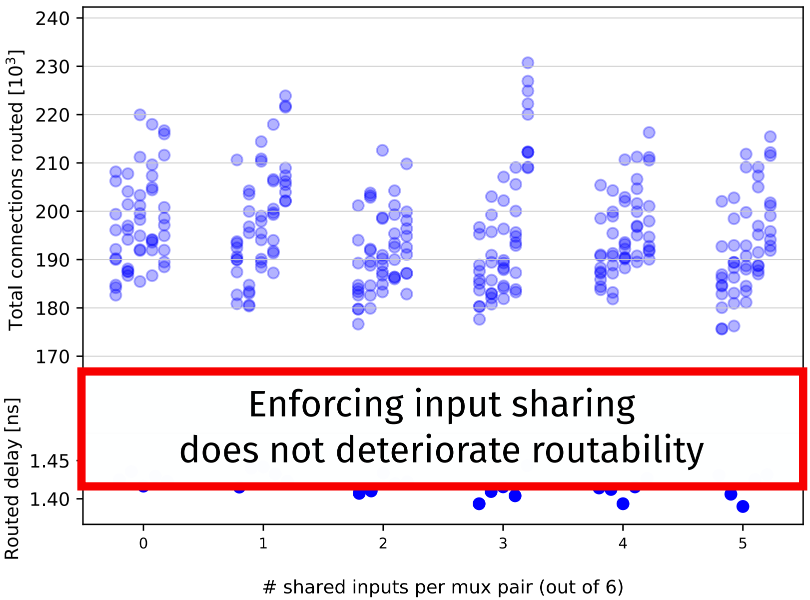
Input sharing: delay



Input sharing: routability



Input sharing: routability



Enforcing input sharing does not deteriorate routability

Other forms of regularity

- Many other forms of regularity presented in the paper
- Arbitrary constraints encodable as ILP supported by the algorithm

Conclusions

- We developed an algorithm that can automatically produce optimized switch-patterns that satisfy arbitrary constraints encodable as ILP

Conclusions

- We developed an algorithm that can automatically produce optimized switch-patterns that satisfy arbitrary constraints encodable as ILP
- Enforcing regularity required for layout
  can sometimes benefit both performance and routability
  and it never significantly deteriorates either

Thank you for attention

https://github.com/EPFL-LAP/fpga23-regularity

If you found the story of Mustafa and Ahmed helpful

Please consider helping people in their region