Automating Design of FPGA Switch-Blocks

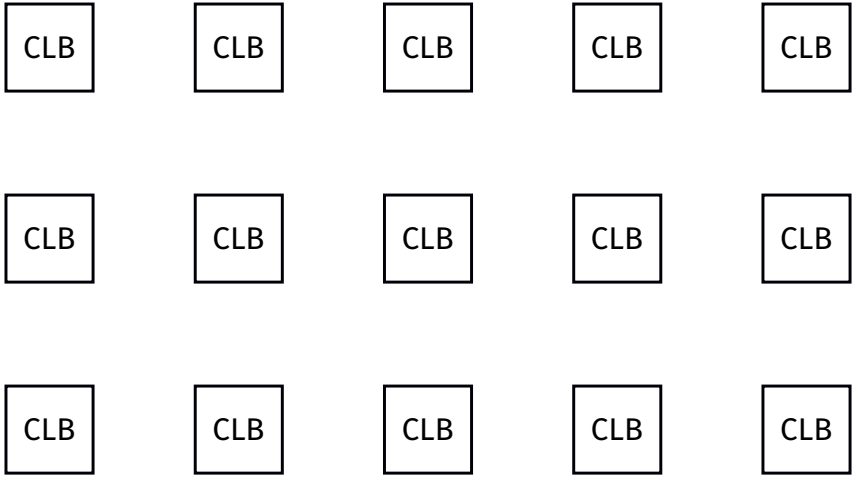
Stefan Nikolić

University of Pennsylvania, 16.02.2023

École Polytechnique Fédérale de Lausanne

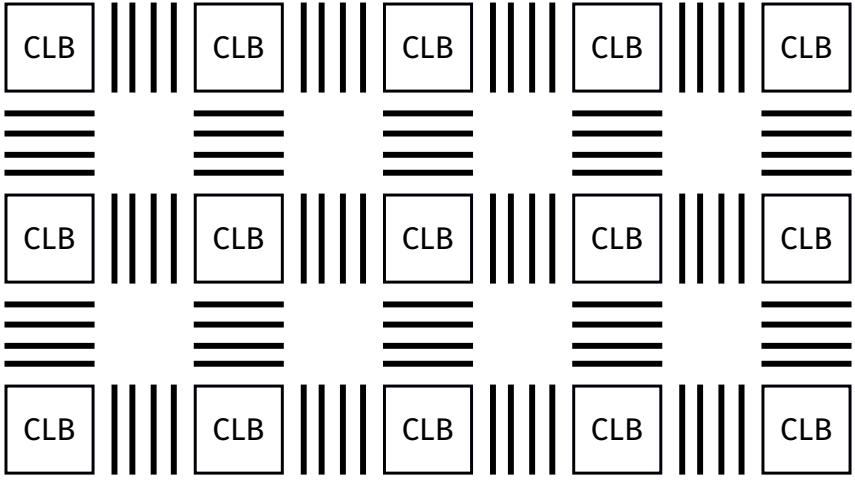
What is the problem?

A very quick review of Island-Style FPGA architecture



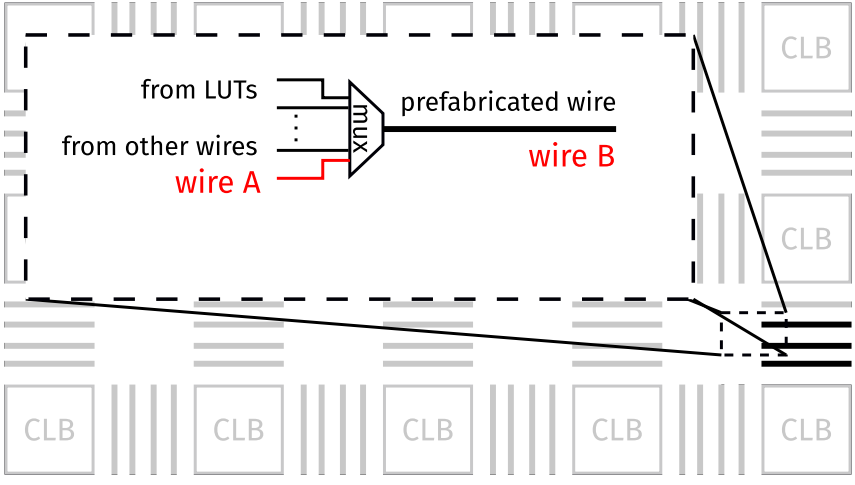
“Islands of LUTs”

A very quick review of Island-Style FPGA architecture



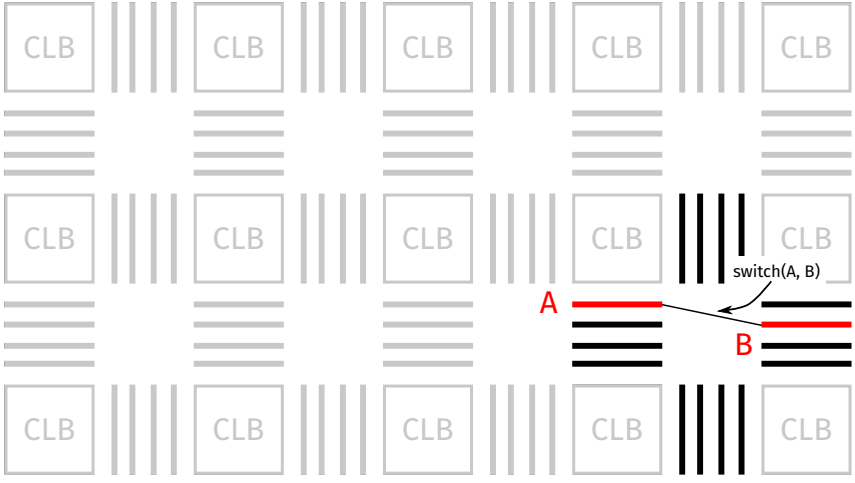
“Surrounded by channels of prefabricated wires”

A very quick review of Island-Style FPGA architecture



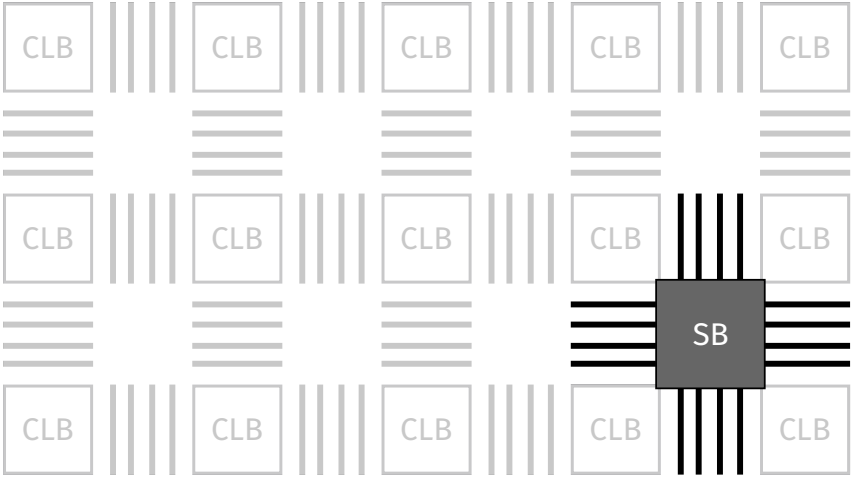
"There is a **SWITCH** between wire **A** and wire **B**"

A very quick review of Island-Style FPGA architecture



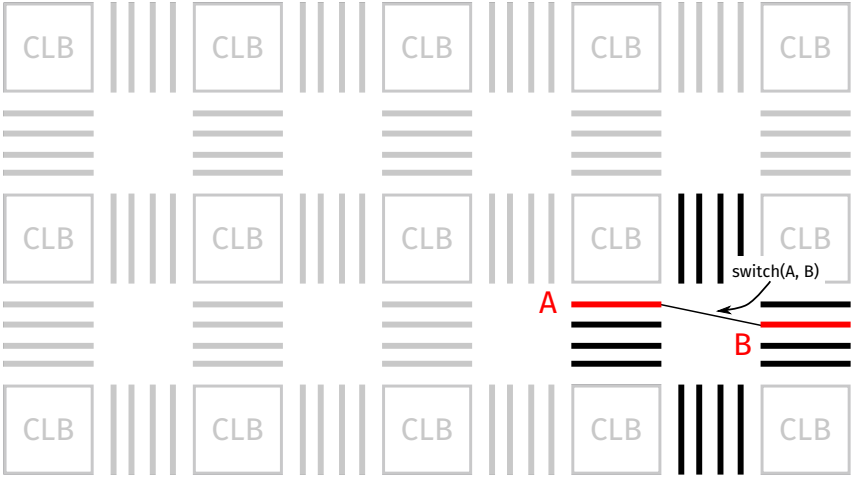
"There is a **SWITCH** between wire **A** and wire **B**"

A very quick review of Island-Style FPGA architecture



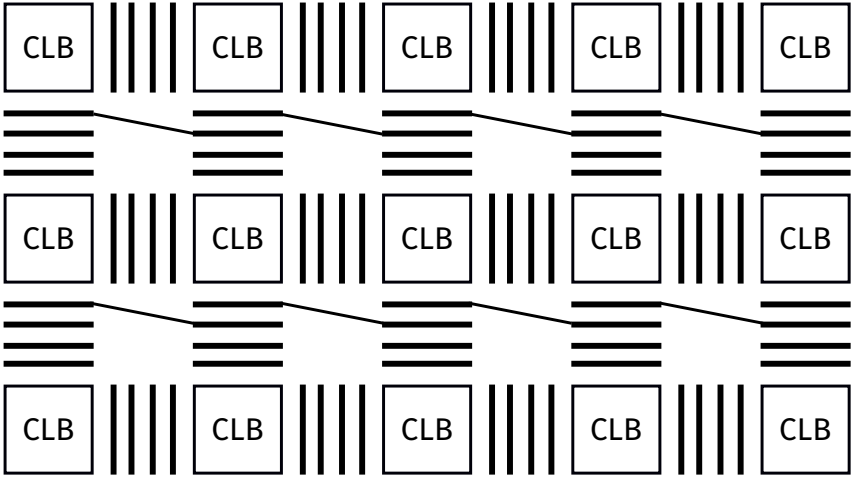
SWITCH-BLOCK (SB)

A very quick review of Island-Style FPGA architecture



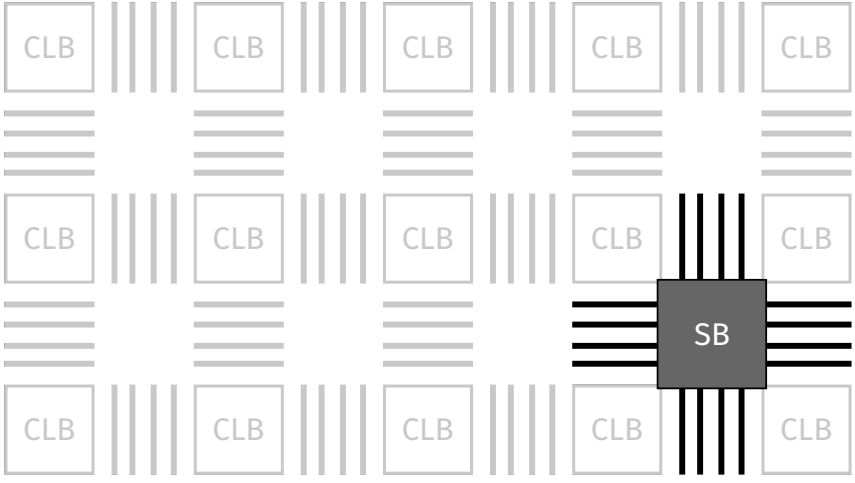
All SBs are identical (e.g., one has switch(A, B) $\Longrightarrow$ all have it)

A very quick review of Island-Style FPGA architecture



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

What is the problem?



Which switches should the SB have?

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. Experim… 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, … In … ble … a mix of Uni… Wilton switch b… ns … o 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

Academic assumptions

VTR 7.0: Next Generation Architecture and CAD System for FPGAs

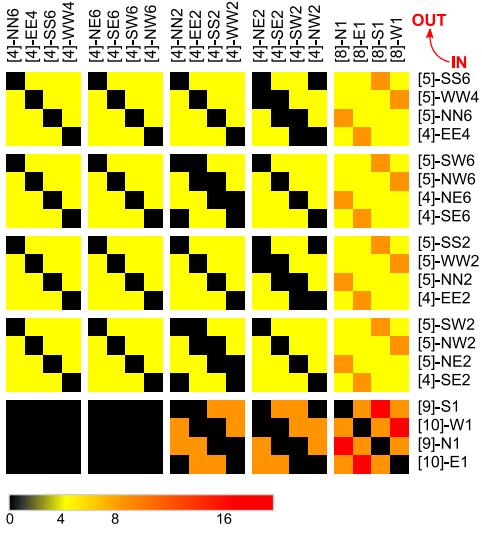
JASON LUU, University of Toronto
JEFFREY GOEDERS, University of British Columbia
MICHAEL WAINBERG, University of Toronto
ANDREW SOMERVILLE, University of New Brunswick
THIEN YU, University of Toronto
KONSTANTIN NASARTSCHUK, University of New Brunswick
MIAD NASR, University of Toronto
SEN WANG, University of New Brunswick
TIM LIU and NOORUDDIN AHMED, University of Toronto
KENNETH B. KENT, University of New Brunswick
JASON ANDERSON, JONATHAN ROSE, and VAUGHN BETZ, University of Toronto

3.1. Comprehensive Architecture File

The *Comprehensive Architecture* is the flagship architecture of the VTR release, and we suggest that new users of VTR begin with it. It describes an FPGA architecture with a number of modern features, including fracturable LUTs, carry-chains, fracturable multipliers, and configurable memories. Its architecture features are chosen to be in line with both the recommendations of prior research and current commercial practice in Xilinx's Virtex 7 and Altera's Stratix IV architectures. The routing architecture uses length-4 wire segments, single-driver routing, an Fc_{in} of 0.15, an Fc_{out} of 0.1, and an Fs of 3.

ACM Transactions on Reconfigurable Technology and Systems, Vol. 7, No. 2, Article 6, Publication date: June 2014.

Industrial reality



H1, H2, H4, H12
V1, V2, V6, V12, V18

Fs = 16

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

Technology driving change

N16: Wu et al.,

"A 16nm FinFET CMOS technology for mobile SoC and computing applications",

IEDM'13

N7: Wu et al.,

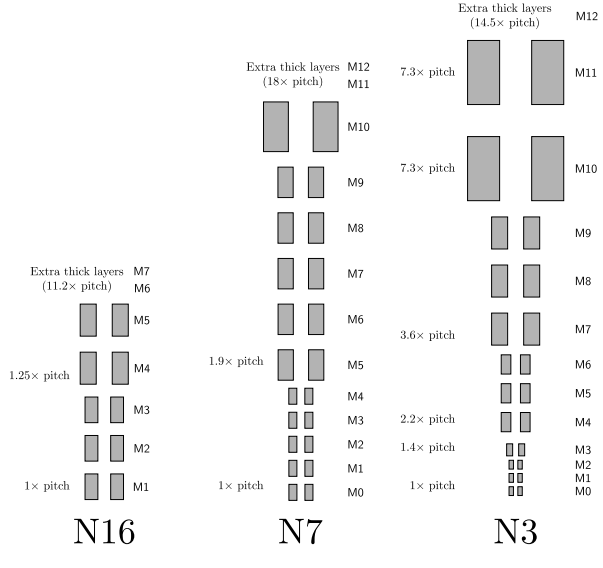
"A 7nm CMOS platform technology featuring 4th generation FinFET transistors with a 0.027 um2 high density 6-T SRAM cell for mobile SoC applications",

IEDM'16

N3: Prasad et al.,

"Buried power rails and back-side power grids: Arm® CPU power delivery network design beyond 5nm",

IEDM'19



[1] Nikolić, Catthoor, Tőkei, and Ienne. Global Is the New Local. FPGA'21

Technology driving change

N16: Wu et al.,

"A 16nm FinFET CMOS technology for mobile SoC and computing applications",

IEDM'13

N7: Wu et al.,

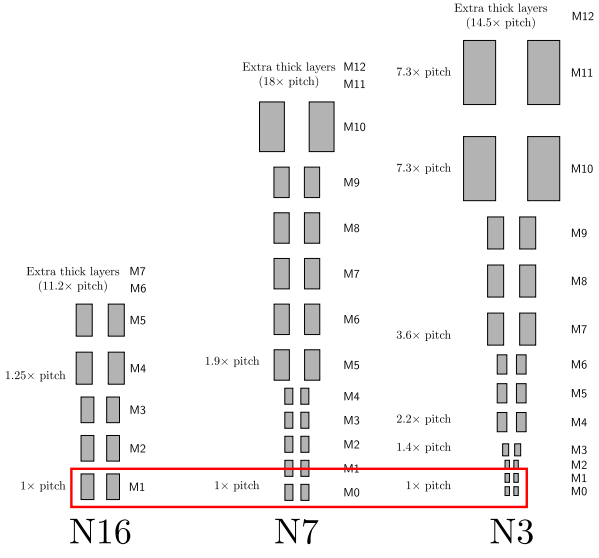
"A 7nm CMOS platform technology featuring 4th generation FinFET transistors with a 0.027 um2 high density 6-T SRAM cell for mobile SoC applications",

IEDM'16

N3: Prasad et al.,

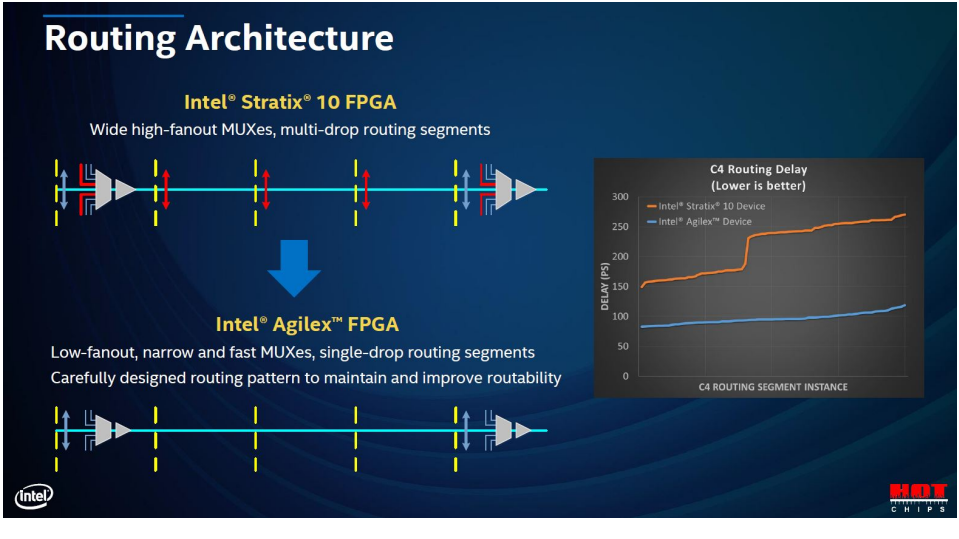
"Buried power rails and back-side power grids: Arm® CPU power delivery network design beyond 5nm",

IEDM'19



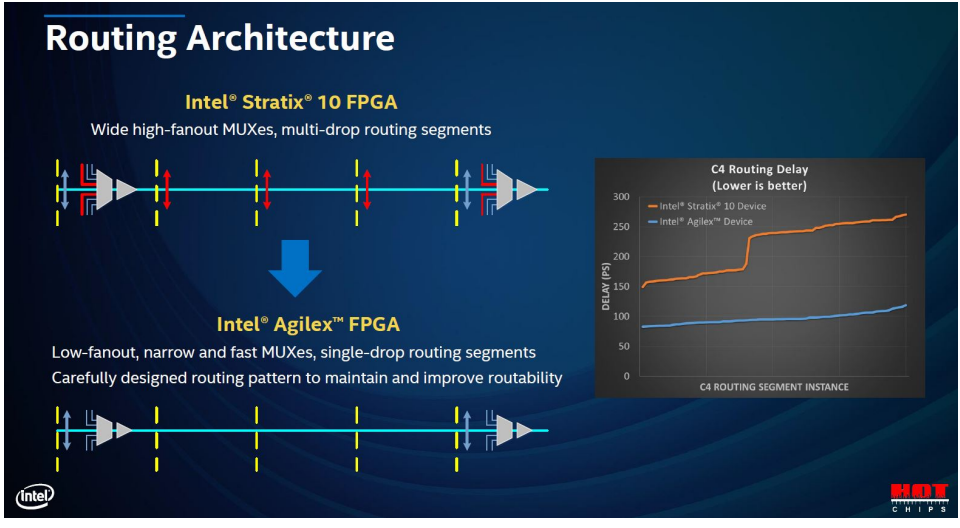
[1] Nikolić, Catthoor, Tőkei, and Ienne. Global Is the New Local. FPGA'21

Technology driving change



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

Technology driving change



"Carefully designed routing pattern to maintain and improve routability"

Two dilemmas

- How to catch up with industry?
- Shouldn’t academia set the pace? (like it did 20 years ago)

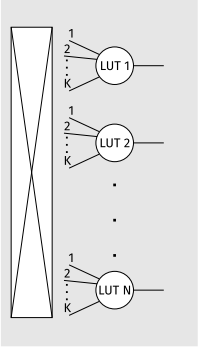
How do we explore FPGA architecture?

A classic approach

The Effect of LUT and Cluster Size on Deep-Submicron FPGA Performance and Density

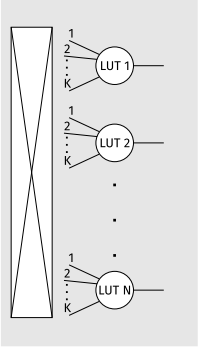
Elias Ahmed and Jonathan Rose

Classical FPGA Architecture Exploration



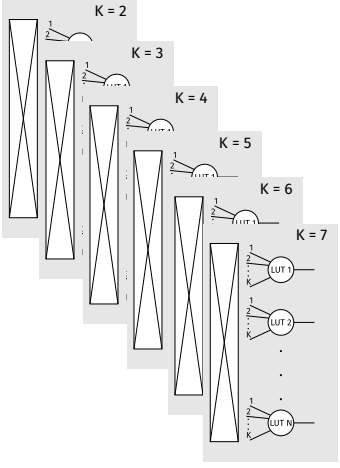
K = ?

Classical FPGA Architecture Exploration



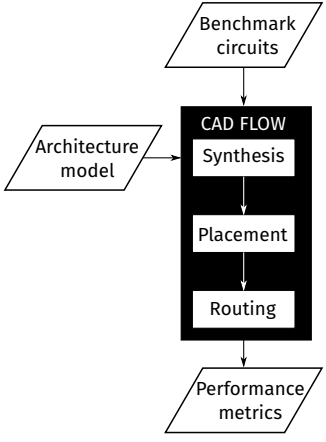
K = ?

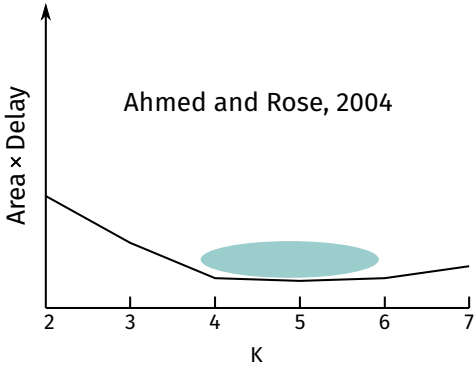
$K \in \{2, 3, 4, 5, 6, 7\}$



Architecture model

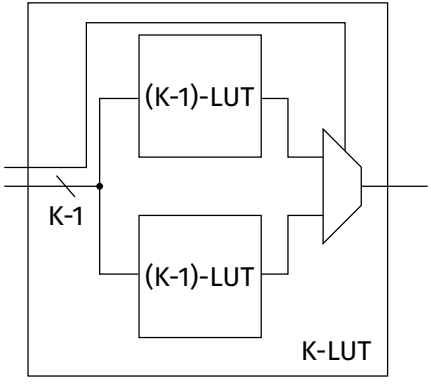
Classical FPGA Architecture Exploration





Ahmed and Rose. The Effect of LUT and Cluster Size on Deep-Submicron FPGA Performance and Density, TVLSI, 2004

What about K > 7?



$$A(K\text{-LUT}) > 2 \times A((K-1)\text{-LUT})$$

Classical FPGA Architecture Exploration

Similar bounds exist for

- cluster size (crossbar area $\Theta(N^2)$)
- channel wires ($t_{L_{max}} < t_{L'} + t_{L''}, \forall L', L'' < L_{max}$)

How did academic research lead the way?

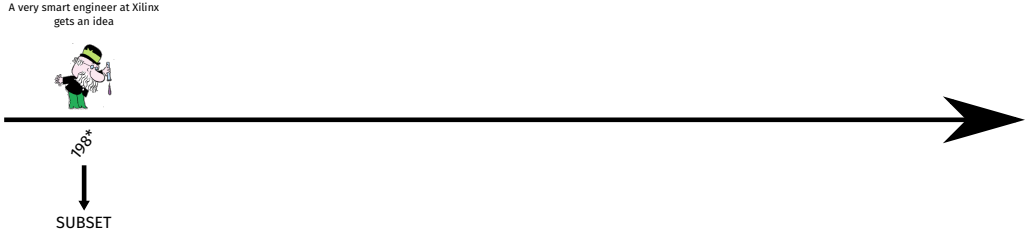
Academic research has been able to quickly explain commercial FPGA architecture design choices and even drive innovation (e.g., Stratix) through **DESIGN SPACE EXPLORATION**

Classical FPGA Architecture Exploration

What about switch-blocks?

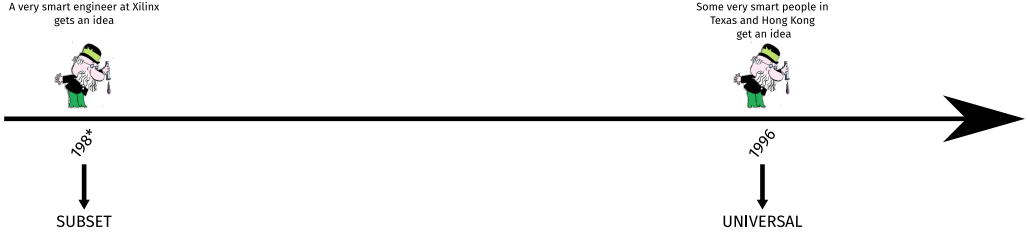
Classical FPGA Architecture Exploration

What about switch-blocks?



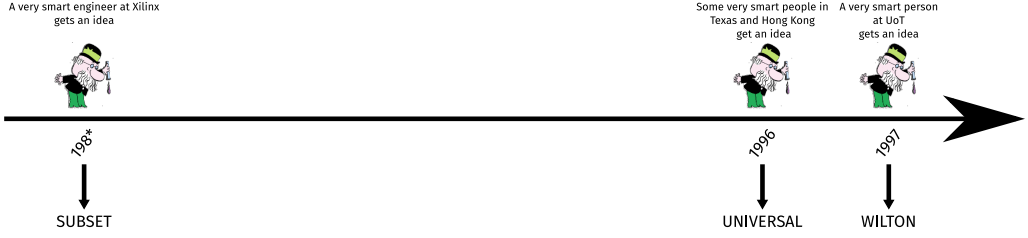
Classical FPGA Architecture Exploration

What about switch-blocks?



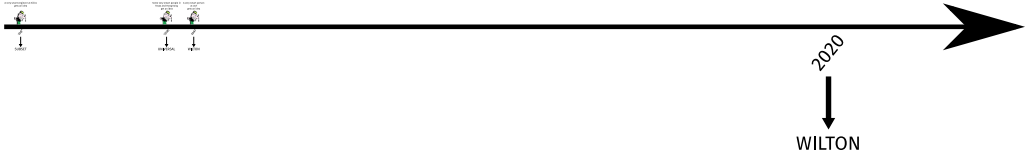
Classical FPGA Architecture Exploration

What about switch-blocks?

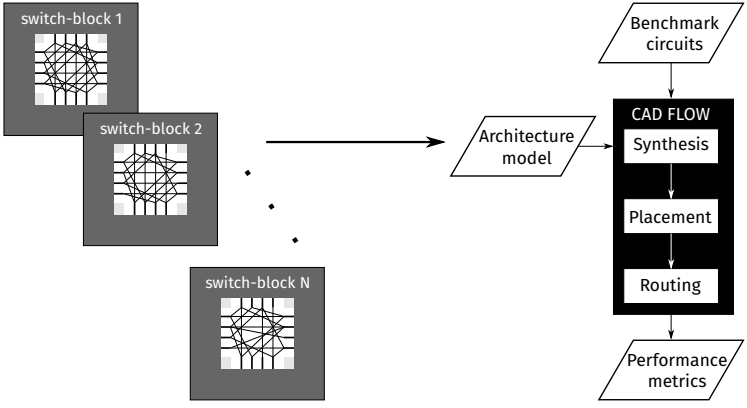


Classical FPGA Architecture Exploration

What about switch-blocks? (apologies for a bit of an exaggeration)



Can’t we automate switch-block exploration too?



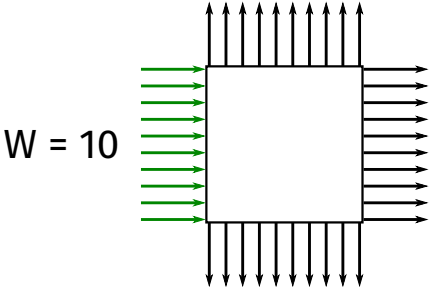
Sure!

Can’t we automate switch-block exploration too?

How big is N?

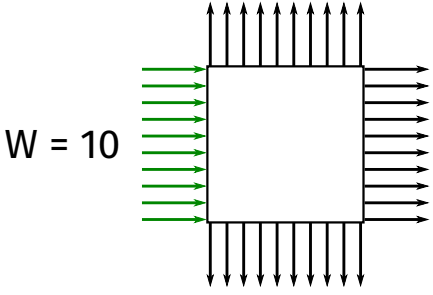
Can’t we automate switch-block exploration too?

How big is N?



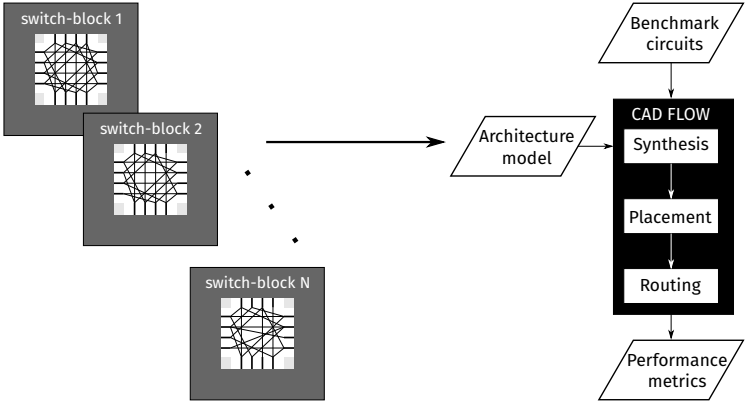
Can’t we automate switch-block exploration too?

How big is N?



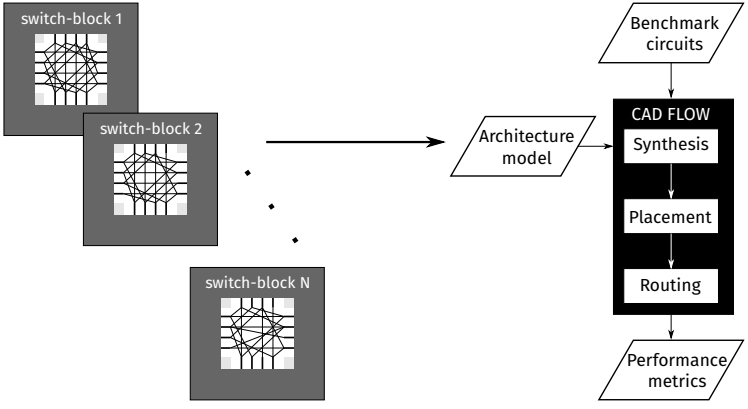
$2^{10 \times 30}$

Can’t we automate switch-block exploration too?



Sure!

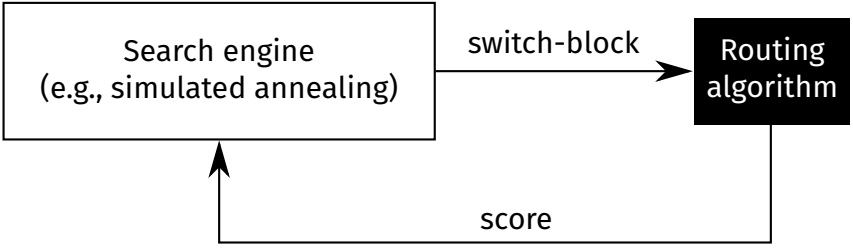
Can’t we automate switch-block exploration too?



Surely not like this!

Avalanche Search

An Intuitive Solution: Iterative Improvement



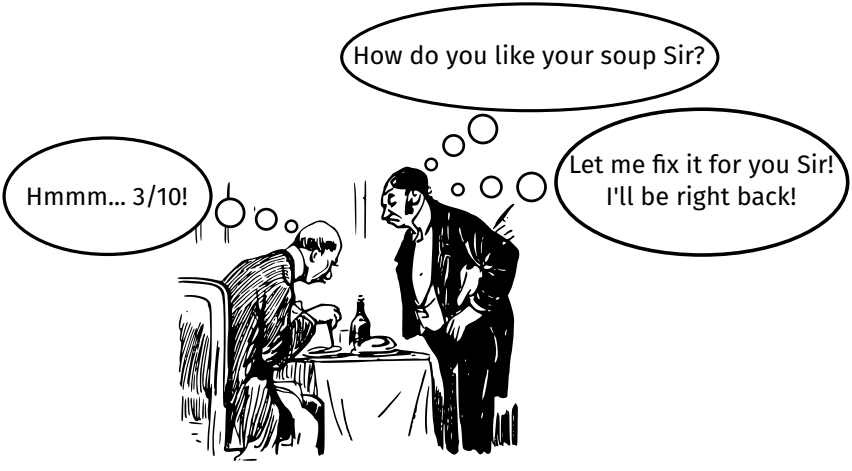
[1] Lin, Wawrzynek, El Gamal. Exploring FPGA routing architecture stochastically. TCAD'10

Using the router as a black box

to evaluate enumerated solutions

is inefficient

Imagine this restaurant service...



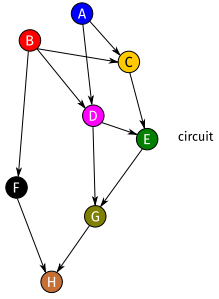
Some things are best left to the consumer

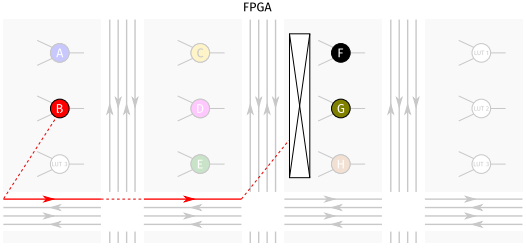
Some things are best left to the consumer

Consumer of switch-block connectivity patterns is the routing algorithm

A very brief review of PathFinder

1: **for** $(u, v) \in E$ **do**
2: path[(u, v)] = shortest_path(u, v)
3: **end for**

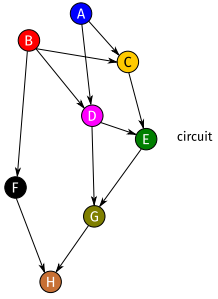


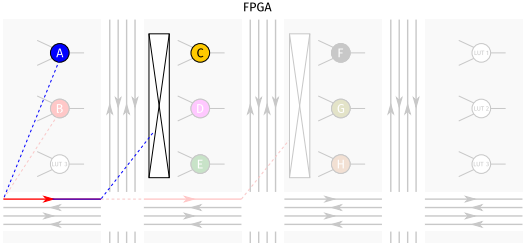


[1] McMurchie and Ebeling. PathFinder: A Negotiation-Based Performance-Driven Router for FPGAs, FPGA'95

A very brief review of PathFinder

1: **for** $(u, v) \in E$ **do**
2: path[(u, v)] = shortest_path(u, v)
3: **end for**

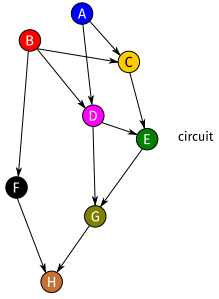


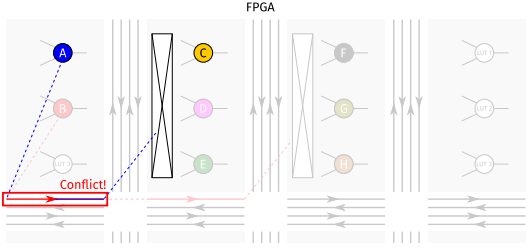


[1] McMurchie and Ebeling. PathFinder: A Negotiation-Based Performance-Driven Router for FPGAs, FPGA'95

A very brief review of PathFinder

1: **for** $(u, v) \in E$ **do**
2: path[(u, v)] = shortest_path(u, v)
3: **end for**

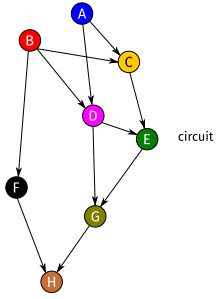


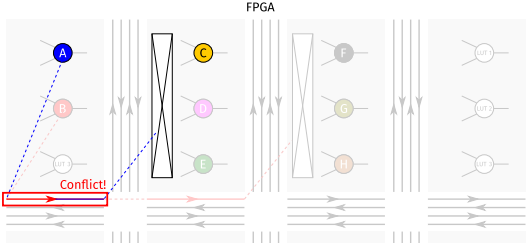


[1] McMurchie and Ebeling. PathFinder: A Negotiation-Based Performance-Driven Router for FPGAs, FPGA'95

A very brief review of PathFinder

1: **for** $(u, v) \in E$ **do**
2: path[(u, v)] = shortest_path(u, v)
3: cost(w) += α, $\forall w \in$ path[(u, v)]
4: **end for**

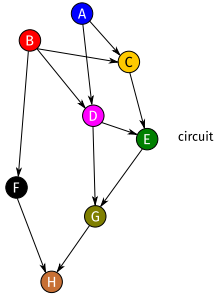


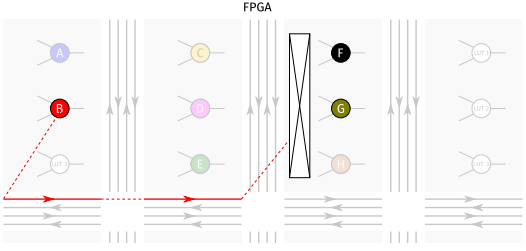


[1] McMurchie and Ebeling. PathFinder: A Negotiation-Based Performance-Driven Router for FPGAs, FPGA'95

A very brief review of PathFinder

1: **for** $(u, v) \in E$ **do**
2: path[(u, v)] = shortest_path(u, v)
3: cost(w) += α, $\forall w \in$ path[(u, v)]
4: **end for**

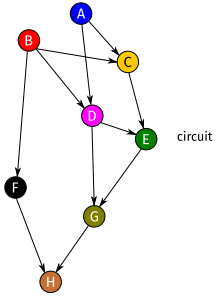


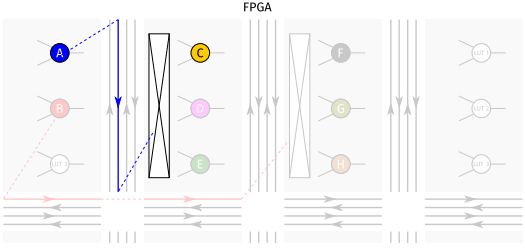


[1] McMurchie and Ebeling. PathFinder: A Negotiation-Based Performance-Driven Router for FPGAs, FPGA'95

A very brief review of PathFinder

1: **for** $(u, v) \in E$ **do**
2: path[(u, v)] = shortest_path(u, v)
3: cost(w) += α, $\forall w \in$ path[(u, v)]
4: **end for**





[1] McMurchie and Ebeling. PathFinder: A Negotiation-Based Performance-Driven Router for FPGAs, FPGA'95

Algorithm 1 Simplified PathFinder [McMurchie95, Betz98]

Input: $G = (V, E)$—rr-graph, $E_c \subseteq V \times V$—all connections to be routed; **Output:** A routing tree of each signal

function CONGESTION_COST(u, s) ▷ computes the congestion cost of node u when routing signal s
  if $u \in RT(s)$ **then return** 0 ▷ if u is already used by one connection of s, it can freely be used by another
  return $b(u) \times (1 + p_{fac} \times O(u)) \times (1 + h_{fac} \times C_h(u))$ ▷ otherwise, account for congestion
for $u \in V$ **do**
  $O(u) = 0; C_h(u) = 0$ ▷ set occupancy and historical congestion of all nodes to 0
  if $(\exists v \in V)\ ((u, v) \in E_c)$ **then**
    $RT(u) = \{u\}$ ▷ initialize the routing tree of each signal
$i = 0; p_{fac} = p_{fac}^{init}$
do
  if $i \geq$ max_iter **then return** UNROUTABLE ▷ no congestion-free routing was found in max_iter iterations
  for $s \in \{u \in V : (\exists v \in V)\ ((u, v) \in E_c)\}$ **do** ▷ all signals are ripped up and rerouted in each iteration;
    ▷ modern incremental routers deviate from this [16]
    for $u \in RT(s)$ **do**
      $O(u) = O(u) - 1$ ▷ reduce the occupancy of all nodes used by the signal s that is ripped up
    $RT(s) = \{s\}; O(s) = O(s) + 1$ ▷ rip up the signal
    for $t \in V : (s, t) \in E_c$ **do**
      $P =$ SHORTEST_PATH$(s, t, \forall u \in V : cong(u) =$ CONGESTION_COST$(u, s))$ ▷ (re)route the connection $s \rightarrow t$
      for $u \in P$ **do**
        if $\neg(u \in RT(s))$ **then**
          $O(u) = O(u) + 1$ ▷ increase the occupancy of all nodes not already used by the signal s
      $RT(s) = RT(s) \cup P$ ▷ add the connection route to the routing tree of s
  for $u \in V$ **do**
    $C_h(u) = C_h(u) + \max(0, O(u) - 1)$ ▷ update historical congestion
  $i = i + 1$
  $p_{fac} = p_{fac} \times p_{fac}^{mult}$ ▷ increase the penalty of using occupied nodes; $p_{fac}^{mult} > 1$ (1.3 is default in VPR [16])
while $\exists u \in V : O(u) > 1$ ▷ finish if there is no congestion
return $\forall RT$ ▷ return all routing trees

Algorithm 1 Simplified PathFinder [McMurchie95, Betz98]

Input: $G = (V, E)$—rr-graph, $E_c \subseteq V \times V$—all connections to be routed; **Output:** A routing tree of each signal

function CONGESTION_COST(u, s) ▷ computes the congestion cost of node u when routing signal s
  if $u \in RT(s)$ **then return** 0 ▷ if u is already used by one connection of s, it can freely be used by another
  return $b(u) \times (1 + p_{fac} \times O(u)) \times (1 + h_{fac} \times C_h(u))$ ▷ otherwise, account for congestion

for $u \in V$ **do**
  $O(u) = 0; C_h(u) = 0$ ▷ set occupancy and historical congestion of all nodes to 0
  if $(\exists v \in V)\ ((u, v) \in E_c)$ **then**
    $RT(u) = \{u\}$ ▷ initialize the routing tree of each signal
$i = 0; p_{fac} = p_{fac}^{init}$
do
  if $i \geq$ max_iter **then return** UNROUTABLE ▷ no congestion-free routing was found in max_iter iterations
  for $s \in \{u \in V : (\exists v \in V)\ ((u, v) \in E_c)\}$ **do** ▷ all signals are ripped up and rerouted in each iteration;
    ▷ modern incremental routers deviate from this [16]
    for $u \in RT(s)$ **do**
      $O(u) = O(u) - 1$ ▷ reduce the occupancy of all nodes used by the signal s that is ripped up
    $RT(s) = \{s\}; O(s) = O(s) + 1$ ▷ rip up the signal
    for $t \in V : (s, t) \in E_c$ **do**
      $P =$ SHORTEST_PATH$(s, t, \forall u \in V : cong(u) =$ CONGESTION_COST$(u, s))$ ▷ (re)route the connection $s \rightarrow t$
      for $u \in P$ **do**
        if $\neg(u \in RT(s))$ **then**
          $O(u) = O(u) + 1$ ▷ increase the occupancy of all nodes not already used by the signal s
      $RT(s) = RT(s) \cup P$ ▷ add the connection route to the routing tree of s
  for $u \in V$ **do**
    $C_h(u) = C_h(u) + \max(0, O(u) - 1)$ ▷ update historical congestion
  $i = i + 1$
  $p_{fac} = p_{fac} \times p_{fac}^{mult}$ ▷ increase the penalty of using occupied nodes; $p_{fac}^{mult} > 1$ (1.3 is default in VPR [16])
while $\exists u \in V : O(u) > 1$ ▷ finish if there is no congestion
return $\forall RT$ ▷ return all routing trees

Algorithm 1 Simplified PathFinder [McMurchie95, Betz98]

Input: $G = (V, E)$—rr-graph, $E_c \subseteq V \times V$—all connections to be routed; **Output:** A routing tree of each signal

function CONGESTION_COST(u, s) ▷ computes the congestion cost of node u when routing signal s
  if $u \in RT(s)$ **then return** 0 ▷ if u is already used by one connection of s, it can freely be used by another
  return $b(u) \times (1 + p_{fac} \times O(u)) \times (1 + h_{fac} \times C_h(u))$ ▷ otherwise, account for congestion

for $u \in V$ **do**
  $O(u) = 0; C_h(u) = 0$ ▷ set occupancy and historical congestion of all nodes to 0
  if $(\exists v \in V)\ ((u, v) \in E_c)$ **then**
    $RT(u) = \{u\}$ ▷ initialize the routing tree of each signal
$i = 0; p_{fac} = p_{fac}^{init}$
do
  if $i \geq$ max_iter **then return** UNROUTABLE ▷ no congestion-free routing was found in max_iter iterations
  for $s \in \{u \in V : (\exists v \in V)\ ((u, v) \in E_c)\}$ **do** ▷ all signals are ripped up and rerouted in each iteration;
    ▷ modern incremental routers deviate from this [16]
    for $u \in RT(s)$ **do**
      $O(u) = O(u) - 1$ ▷ reduce the occupancy of all nodes used by the signal s that is ripped up
    $RT(s) = \{s\}; O(s) = O(s) + 1$ ▷ rip up the signal
    for $t \in V : (s, t) \in E_c$ **do**
      $P =$ SHORTEST_PATH$(s, t, \forall u \in V : cong(u) =$ CONGESTION_COST$(u, s))$ ▷ (re)route the connection $s \rightarrow t$
      for $u \in P$ **do**
        if $\neg(u \in RT(s))$ **then**
          $O(u) = O(u) + 1$ ▷ increase the occupancy of all nodes not already used by the signal s
      $RT(s) = RT(s) \cup P$ ▷ add the connection route to the routing tree of s
  for $u \in V$ **do**
    $C_h(u) = C_h(u) + \max(0, O(u) - 1)$ ▷ update historical congestion
  $i = i + 1$
  $p_{fac} = p_{fac} \times p_{fac}^{mult}$ ▷ increase the penalty of using occupied nodes; $p_{fac}^{mult} > 1$ (1.3 is default in VPR [16])
while $\exists u \in V : O(u) > 1$ ▷ finish if there is no congestion
return $\forall RT$ ▷ return all routing trees

Algorithm 1 Simplified PathFinder [McMurchie95, Betz98]

Input: $G = (V, E)$—rr-graph, $E_c \subseteq V \times V$—all connections to be routed; **Output:** A routing tree of each signal

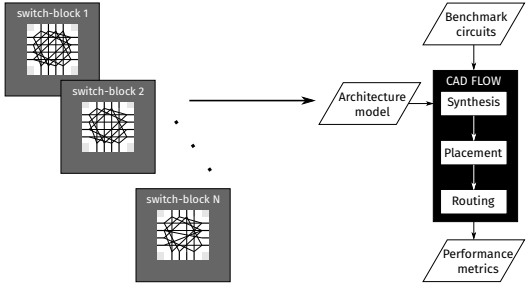
function CONGESTION_COST(u, s) ▷ computes the congestion cost of node u when routing signal s
  if $u \in RT(s)$ **then return** 0 ▷ if u is already used by one connection of s, it can freely be used by another
  return $b(u) \times (1 + p_{fac} \times O(u)) \times (1 + h_{fac} \times C_h(u))$ ▷ otherwise, account for congestion
for $u \in V$ **do**
  $O(u) = 0; C_h(u) = 0$ ▷ set occupancy and historical congestion of all nodes to 0
  if $(\exists v \in V)\ ((u, v) \in E_c)$ **then**
    $RT(u) = \{u\}$ ▷ initialize the routing tree of each signal
$i = 0; p_{fac} = p_{fac}^{init}$
do
  if $i \geq$ max_iter **then return** UNROUTABLE ▷ no congestion-free routing was found in max_iter iterations
  for $s \in \{u \in V : (\exists v \in V)\ ((u, v) \in E_c)\}$ **do** ▷ all signals are ripped up and rerouted in each iteration;
    ▷ modern incremental routers deviate from this [16]
    for $u \in RT(s)$ **do**
      $O(u) = O(u) - 1$ ▷ reduce the occupancy of all nodes used by the signal s that is ripped up
    $RT(s) = \{s\}; O(s) = O(s) + 1$ ▷ rip up the signal
    for $t \in V : (s, t) \in E_c$ **do**
      $P =$ SHORTEST_PATH$(s, t, \forall u \in V : cong(u) =$ CONGESTION_COST$(u, s))$ ▷ (re)route the connection $s \longrightarrow t$
      for $u \in P$ **do**
        if $\neg(u \in RT(s))$ **then**
          $O(u) = O(u) + 1$ ▷ increase the occupancy of all nodes not already used by the signal s
      $RT(s) = RT(s) \cup P$ ▷ add the connection route to the routing tree of s
  for $u \in V$ **do**
    $C_h(u) = C_h(u) + \max(0, O(u) - 1)$ ▷ update historical congestion
  $i = i + 1$
  $p_{fac} = p_{fac} \times p_{fac}^{mult}$ ▷ increase the penalty of using occupied nodes; $p_{fac}^{mult} > 1$ (1.3 is default in VPR [16])
while $\exists u \in V : O(u) > 1$ ▷ finish if there is no congestion
return $\forall RT$ ▷ return all routing trees

Algorithm 1 Simplified PathFinder [McMurchie95, Betz98]

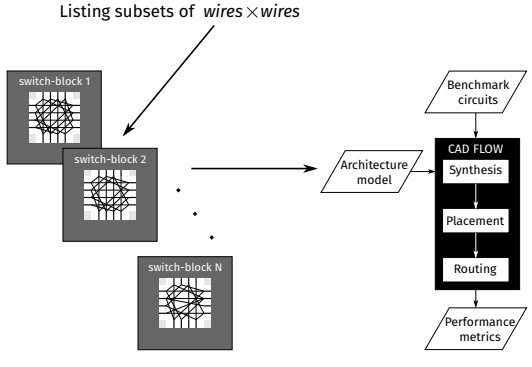
Input: $G = (V, E)$—rr-graph, $E_c \subseteq V \times V$—all connections to be routed; **Output:** A routing tree of each signal

function CONGESTION_COST(u, s) ▷ computes the congestion cost of node u when routing signal s
  if $u \in RT(s)$ **then return** 0 ▷ if u is already used by one connection of s, it can freely be used by another
  return $b(u) \times (1 + p_{fac} \times O(u)) \times (1 + h_{fac} \times C_h(u))$ ▷ otherwise, account for congestion
for $u \in V$ **do**
  $O(u) = 0; C_h(u) = 0$ ▷ set occupancy and historical congestion of all nodes to 0
  if $(\exists v \in V)\ ((u, v) \in E_c)$ **then**
    $RT(u) = \{u\}$ ▷ initialize the routing tree of each signal
$i = 0; p_{fac} = p_{fac}^{init}$
do
  if $i \geq$ max_iter **then return** UNROUTABLE ▷ no congestion-free routing was found in max_iter iterations
  for $s \in \{u \in V : (\exists v \in V)\ ((u, v) \in E_c)\}$ **do** ▷ all signals are ripped up and rerouted in each iteration;
    ▷ modern incremental routers deviate from this [16]
    for $u \in RT(s)$ **do**
      $O(u) = O(u) - 1$ ▷ reduce the occupancy of all nodes used by the signal s that is ripped up
    $RT(s) = \{s\}; O(s) = O(s) + 1$ ▷ rip up the signal
    for $t \in V : (s, t) \in E_c$ **do**
      $P =$ SHORTEST_PATH$(s, t, \forall u \in V : cong(u) =$ CONGESTION_COST$(u, s))$ ▷ (re)route the connection $s \rightarrow t$
      for $u \in P$ **do**
        if $\neg(u \in RT(s))$ **then**
          $O(u) = O(u) + 1$ ▷ increase the occupancy of all nodes not already used by the signal s
      $RT(s) = RT(s) \cup P$ ▷ add the connection route to the routing tree of s
  for $u \in V$ **do**
    $C_h(u) = C_h(u) + \max(0, O(u) - 1)$ ▷ update historical congestion
  $i = i + 1$
  $p_{fac} = p_{fac} \times p_{fac}^{mult}$ ▷ increase the penalty of using occupied nodes; $p_{fac}^{mult} > 1$ (1.3 is default in VPR [16])
while $\exists u \in V : O(u) > 1$ ▷ finish if there is no congestion
return $\forall RT$ ▷ return all routing trees

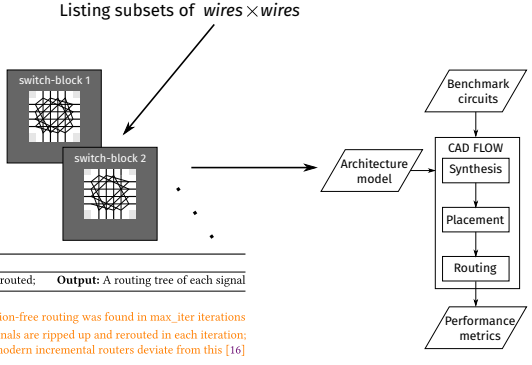
How to avoid listing individual solutions?



How to avoid listing individual solutions?



How to avoid listing individual solutions?

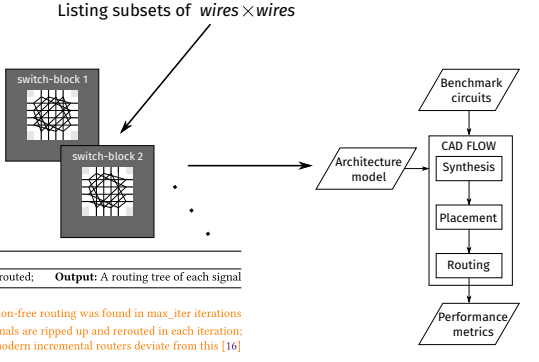


Algorithm 1 Simplified PathFinder [McMurchie95, Betz98]

Input: $G = (V, E)$—rr-graph, $E_c \subseteq V \times V$—all connections to be routed; **Output:** A routing tree of each signal

```
do
    if i ≥ max_iter then return UNROUTABLE        ▷ no congestion-free routing was found in max_iter iterations
    for s ∈ {u ∈ V : (∃v ∈ V) ((u, v) ∈ E_c)} do   ▷ all signals are ripped up and rerouted in each iteration;
                                                  ▷ modern incremental routers deviate from this [16]
        for u ∈ RT(s) do
            O(u) = O(u) − 1                       ▷ reduce the occupancy of all nodes used by the signal s that is ripped up
        RT(s) = {s}; O(s) = O(s) + 1              ▷ rip up the signal
        for t ∈ V : (s, t) ∈ E_c do
            P = SHORTEST_PATH(s, t, ∀u ∈ V : cong(u) = CONGESTION_COST(u, s))   ▷ (re)route the connection s → t
            for u ∈ P do
                if ¬(u ∈ RT(s)) then
                    O(u) = O(u) + 1               ▷ increase the occupancy of all nodes not already used by the signal s
            RT(s) = RT(s) ∪ P                     ▷ add the connection route to the routing tree of s
    for u ∈ V do
        C_h(u) = C_h(u) + max(0, O(u) − 1)        ▷ update historical congestion
    i = i + 1
    p_fac = p_fac × p_fac^mult                    ▷ increase the penalty of using occupied nodes; p_fac^mult > 1 (1.3 is default in VPR [16])
while ∃u ∈ V : O(u) > 1                           ▷ finish if there is no congestion
return ∀RT                                        ▷ return all routing trees
```

How to avoid listing individual solutions?



Algorithm 1 Simplified PathFinder **[McMurchie95, Betz98]**

Input: $G = (V, E)$—rr-graph, $E_c \subseteq V \times V$—all connections to be routed; **Output:** A routing tree of each signal

```
do
    if i ≥ max_iter then return UNROUTABLE        ▷ no congestion-free routing was found in max_iter iterations
    for s ∈ {u ∈ V : (∃v ∈ V) ((u, v) ∈ E_c)} do   ▷ all signals are ripped up and rerouted in each iteration;
                                                  ▷ modern incremental routers deviate from this [16]
        for u ∈ RT(s) do
            O(u) = O(u) − 1                       ▷ reduce the occupancy of all nodes used by the signal s that is ripped up
        RT(s) = {s}; O(s) = O(s) + 1
        for t ∈ V : (s, t) ∈ E_c do
            P = SHORTEST_PATH(s, t, ∀u ∈ V : cong(u) = CONGESTION_COST(u, s))   ▷ (re)r
            for u ∈ P do
                if ¬(u ∈ RT(s)) then
                    O(u) = O(u) + 1               ▷ increase the occupancy of all nodes not a
                RT(s) = RT(s) ∪ P                 ▷ add the connection ro
    for u ∈ V do
        C_h(u) = C_h(u) + max(0, O(u) − 1)        ▷ up
    i = i + 1
    p_fac = p_fac × p_fac^mult                    ▷ increase the penalty of using occupied nodes; p_fac^mult > 1
while ∃u ∈ V : O(u) > 1                           ▷ finis... is there is no congestion
return ∀RT                                        ▷ return all routing trees
```

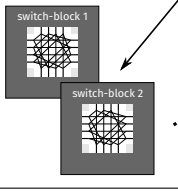
The more paths use a switch

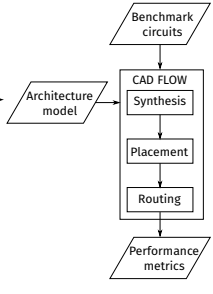
$s \in SB \subset wires \times wires$

the more useful it is in the architecture

How to avoid listing individual solutions?

~~Listing subsets of wires×wires~~





Algorithm 1 Simplified PathFinder **[McMurchie95, Betz98]**

Input: $G = (V, E)$—rr-graph, $E_c \subseteq V \times V$—all connections to be routed; **Output:** A routing tree of each signal

do
  if $i \geq$ max_iter **then return** UNROUTABLE ▷ no congestion-free routing was found in max_iter iterations
  for $s \in \{u \in V : (\exists v \in V)\ ((u, v) \in E_c)\}$ **do** ▷ all signals are ripped up and rerouted in each iteration;
    ▷ modern incremental routers deviate from this [16]
    for $u \in RT(s)$ **do**
      $O(u) = O(u) - 1$ ▷ reduce the occupancy of all nodes used by the signal s that is ripped up
    $RT(s) = \{s\}$; $O(s) = O(s) + 1$
    for $t \in V : (s, t) \in E_c$ **do**
      $P =$ SHORTEST_PATH$(s, t, \forall u \in V : cong(u) =$ CONGESTION_COST$(u, s))$ ▷ (re)r
      for $u \in P$ **do**
        if $\neg(u \in RT(s))$ **then**
          $O(u) = O(u) + 1$ ▷ increase the occupancy of all nodes not a
        $RT(s) = RT(s) \cup P$ ▷ add the connection ro
    for $u \in V$ **do**
      $C_h(u) = C_h(u) + \max(0, O(u) - 1)$ ▷ up
    $i = i + 1$
    $p_{fac} = p_{fac} \times p_{fac}^{mult}$ ▷ increase the penalty of using occupied nodes; $p_{fac}^{mult} > 1$
while $\exists u \in V : O(u) > 1$ ▷ finis...
return $\forall RT$ ▷ return all routing trees

Represent all possible switches

The more paths use a switch

$s \in SB = wires \times wires$

the more useful it is in the architecture

Construct the solution from the most-used ones

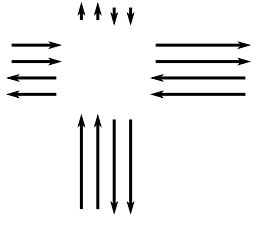
Avalanche Search [Nikolić and Ienne. Turning PathFinder Upside-Down. FPL'21]

1. switches marked for fabrication $\leftarrow \{\}$
2. all possible switches can be used at cost C

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

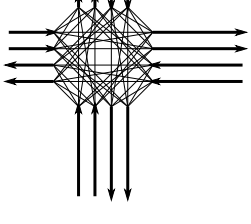
switches marked for fabrication

discounted cost
(0)



switches available to the router

everything else
is expensive



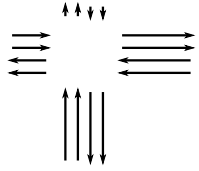
Avalanche Search [Nikolić and Ienne. Turning PathFinder Upside-Down. FPL'21]

1. switches marked for fabrication $\leftarrow \{\}$
2. all possible switches can be used at cost C
3. let PathFinder route the circuits

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

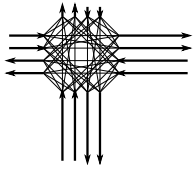
switches marked for fabrication

discounted cost
(0)

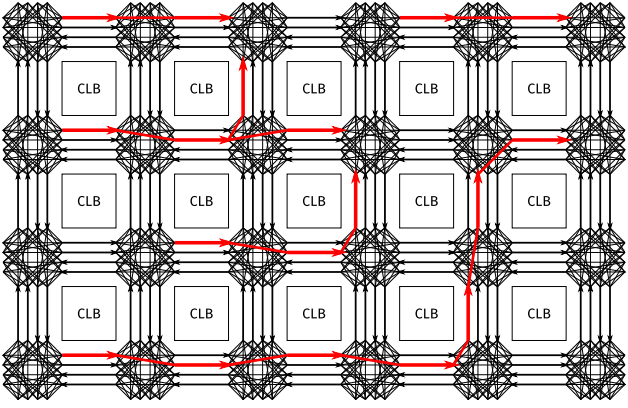


switches available to the router

everything else
is expensive



PathFinder routes circuits (each connection by a shortest path)

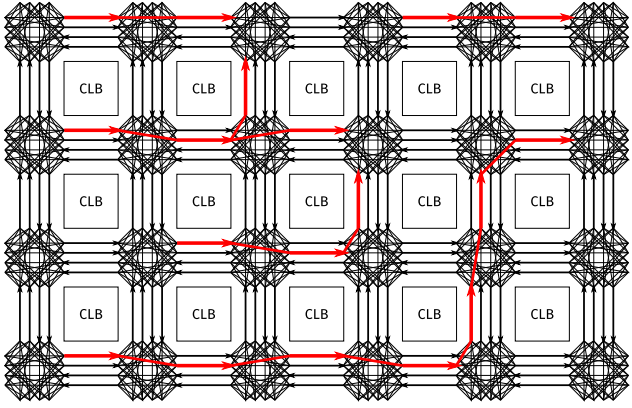


Avalanche Search [Nikolić and Ienne. Turning PathFinder Upside-Down. FPL'21]

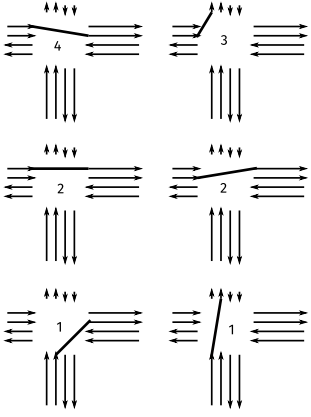
1. switches marked for fabrication $\leftarrow \{\}$
2. all possible switches can be used at cost C
3. let PathFinder route the circuits
4. if no unmarked switches are used, done

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

PathFinder routes circuits (each connection by a shortest path)



Count in how many switch-blocks each switch was used

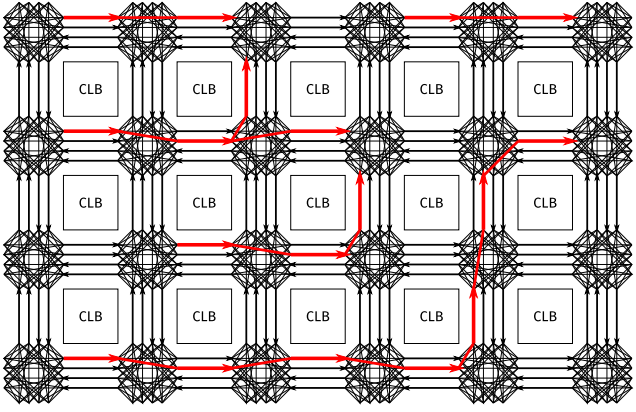


Avalanche Search [Nikolić and Ienne. Turning PathFinder Upside-Down. FPL'21]

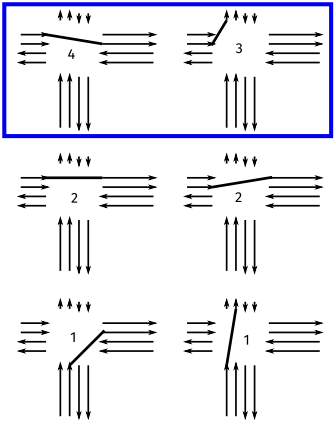
1. switches marked for fabrication $\leftarrow \{\}$
2. all possible switches can be used at cost C
3. let PathFinder route the circuits
4. if no unmarked switches are used, done
5. mark n most-used unmarked switches and set their cost to 0

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

PathFinder routes circuits (each connection by a shortest path)



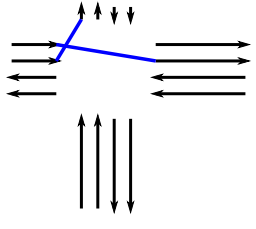
Give discounts and mark for fabrication



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

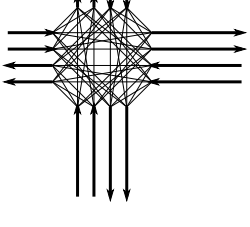
switches marked for fabrication

discounted cost
(0)



switches available to the router

everything else
is expensive



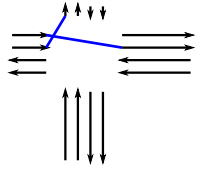
Avalanche Search [Nikolić and Ienne. Turning PathFinder Upside-Down. FPL'21]

1. switches marked for fabrication $\leftarrow \{\}$
2. all possible switches can be used at cost C
3. let PathFinder route the circuits
4. if no unmarked switches are used, done
5. mark n most-used unmarked switches and set their cost to 0
6. goto 3

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

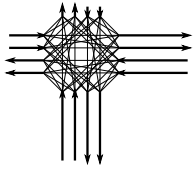
switches marked for fabrication

discounted cost
(0)

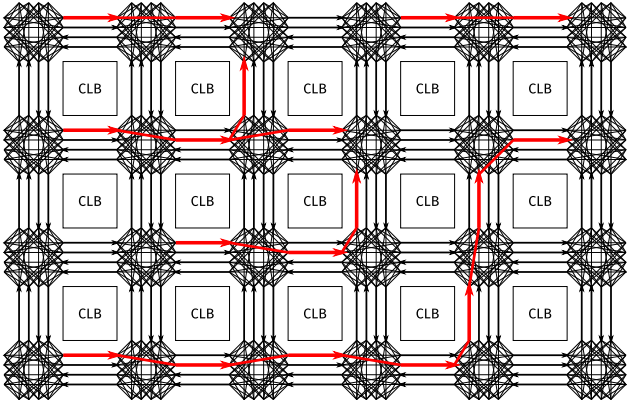


switches available to the router
(Mustafa's catalogue)

everything else
is expensive



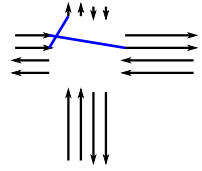
PathFinder routes circuits (each connection by a shortest path)



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

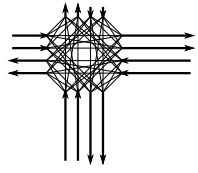
switches marked for fabrication

discounted cost
(0)

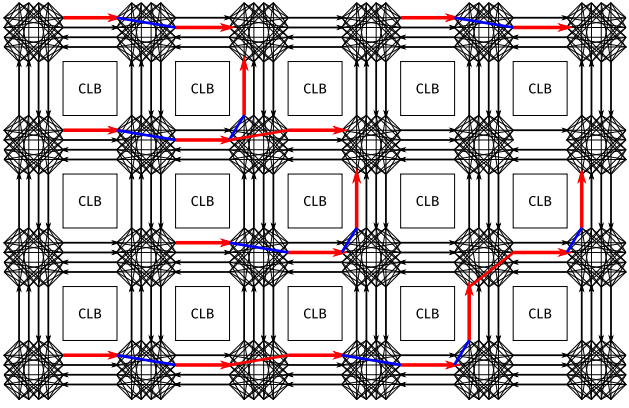


switches available to the router
(Mustafa's catalogue)

everything else
is expensive



PathFinder routes circuits (each connection by a shortest path)

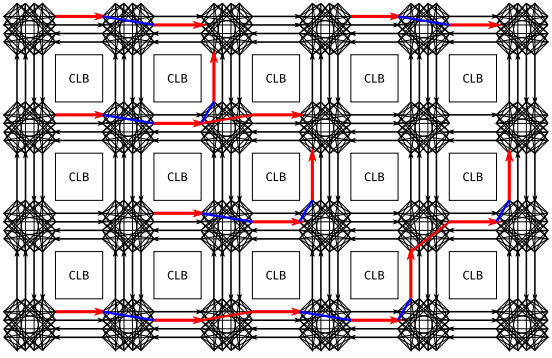


Avalanche Search [Nikolić and Ienne. Turning PathFinder Upside-Down. FPL'21]

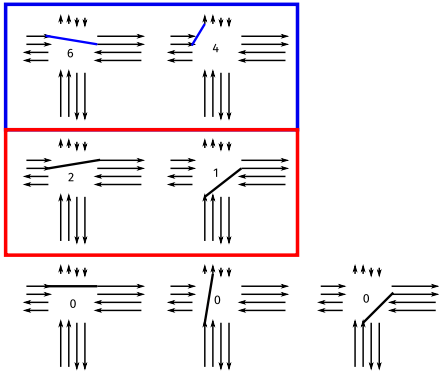
1. switches marked for fabrication $\leftarrow \{\}$
2. all possible switches can be used at cost C
3. let PathFinder route the circuits
4. if no unmarked switches are used, done

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

PathFinder routes circuits (each connection by a shortest path)



Count in how many switch-blocks each switch was used

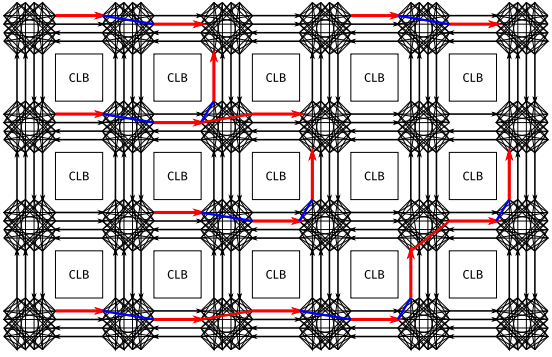


Avalanche Search [Nikolić and Ienne. Turning PathFinder Upside-Down. FPL'21]

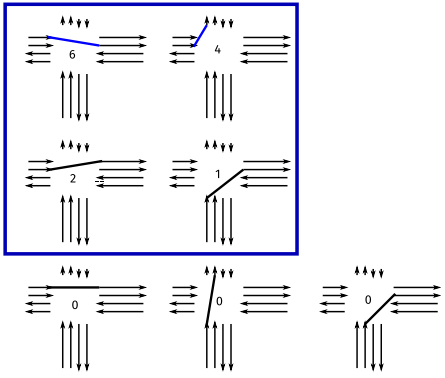
1. switches marked for fabrication $\leftarrow \{\}$
2. all possible switches can be used at cost C
3. let PathFinder route the circuits
4. if no unmarked switches are used, done
5. mark n most-used unmarked switches and set their cost to 0

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

PathFinder routes circuits (each connection by a shortest path)



Give discounts and mark for fabrication



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
4. if no unmarked switches are used, done
5. mark n most-used unmarked switches and set their cost to 0
6. goto 3

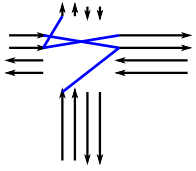
Avalanche Search [Nikolić and Ienne. Turning PathFinder Upside-Down. FPL'21]

1. switches marked for fabrication $\leftarrow \{\}$
2. all possible switches can be used at cost C
3. let PathFinder route the circuits

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

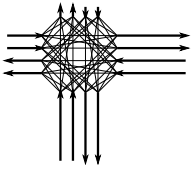
switches marked for fabrication

discounted cost
(0)

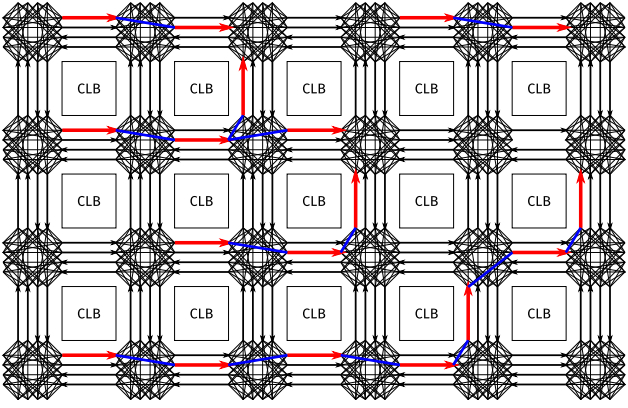


switches available to the router
(Mustafa's catalogue)

everything else
is expensive



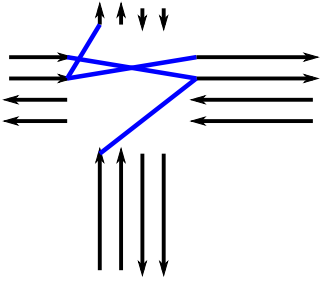
PathFinder routes circuits (each connection by a shortest path)



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
4. if no unmarked switches are used, done

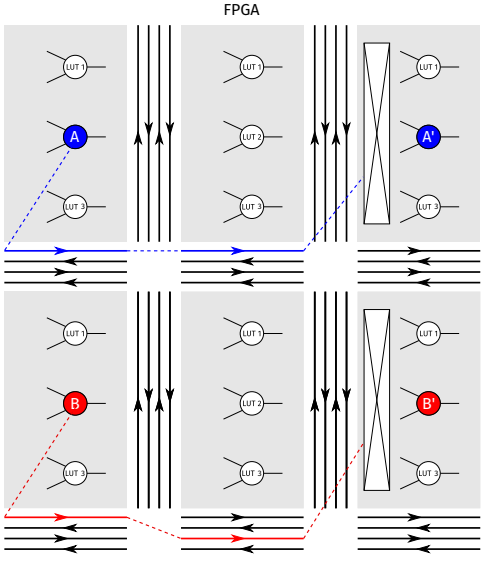
Avalanche Search [Nikolić and Ienne. Turning PathFinder Upside-Down. FPL'21]



Final switch-pattern

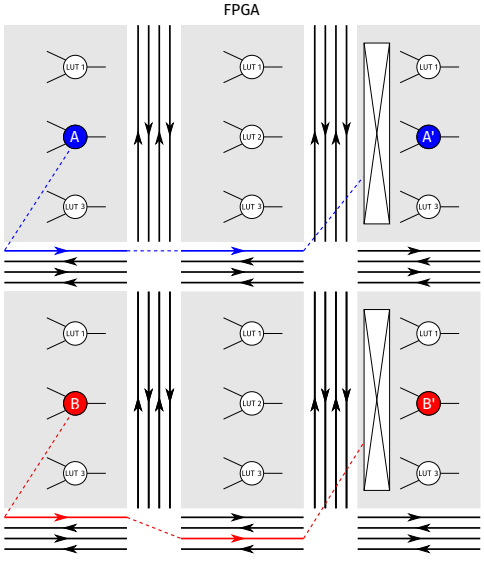
Where is the “avalanche”?

Avalanche costs



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

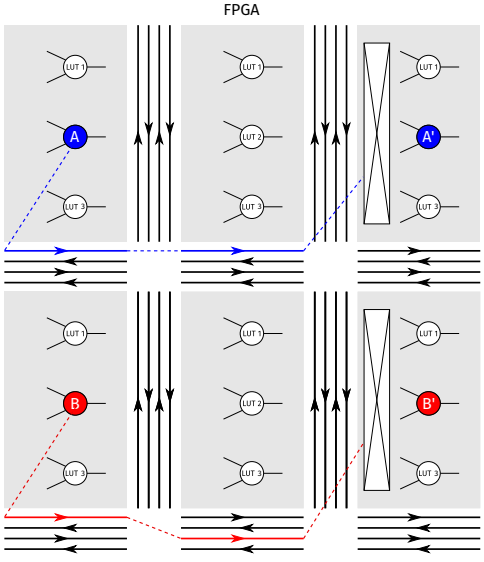
Avalanche costs



No congestion ✓

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

Avalanche costs



No congestion ✓

But, switch-block must include
both the straight and the slanted switch
where one may suffice

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

How does PathFinder achieve signal spreading?

Congestion cost:

- all wire **INSTANCES** (individual rr-graph nodes) are initially **CHEAP**
- their cost **INCREASES** "in proportion with" conflict magnitude

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

How does PathFinder achieve signal spreading?

We need concentration instead of spreading

$\Longrightarrow$ apply the same principle in reverse

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

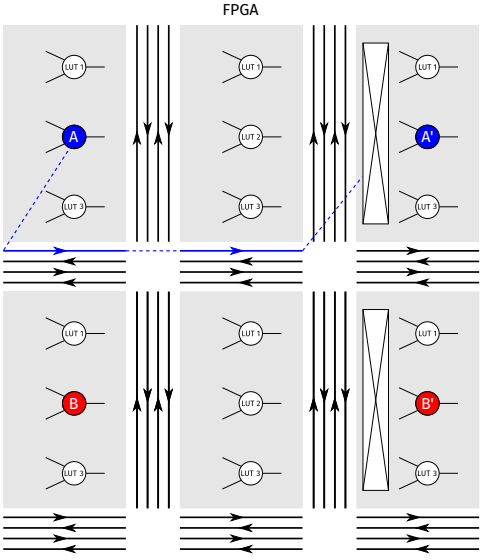
Avalanche costs

Avalanche cost:

- all switches are initially **EXPENSIVE**
- their cost **DROPS** in proportion with how many of their instances are used

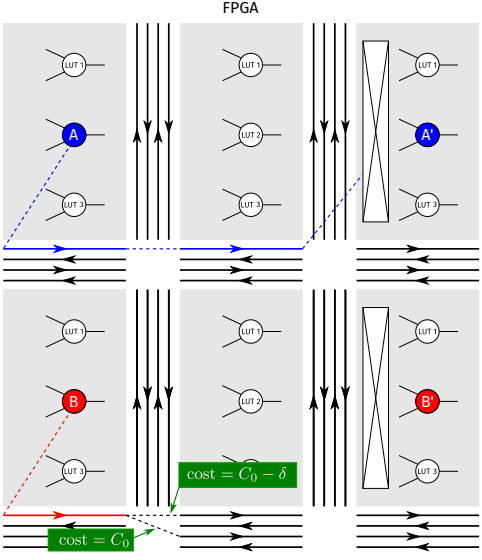
[1] Nikolić and Ienne. Turning PathFinder Upside Down, FPL'21, **Best Paper Award**

Avalanche costs



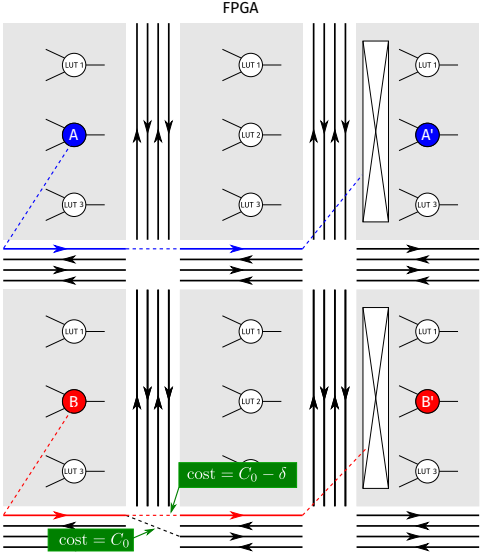
[1] Nikolić and Ienne. Turning PathFinder Upside Down, FPL'21, **Best Paper Award**

Avalanche costs



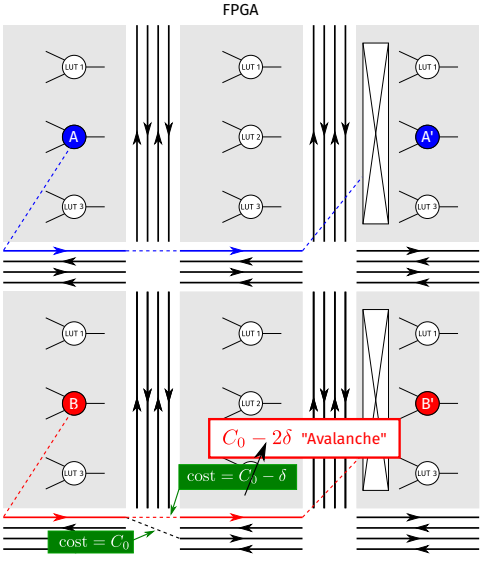
[1] Nikolić and Ienne. Turning PathFinder Upside Down, FPL'21, **Best Paper Award**

Avalanche costs



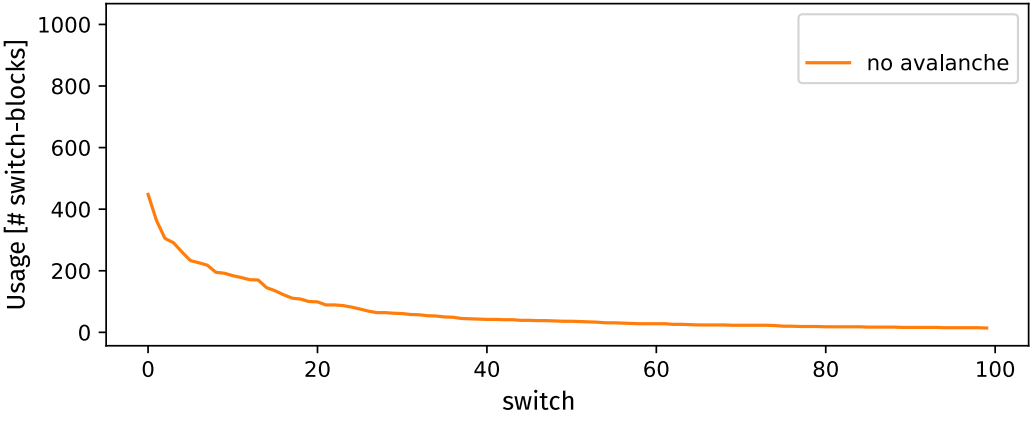
[1] Nikolić and Ienne. Turning PathFinder Upside Down, FPL'21, **Best Paper Award**

Avalanche costs



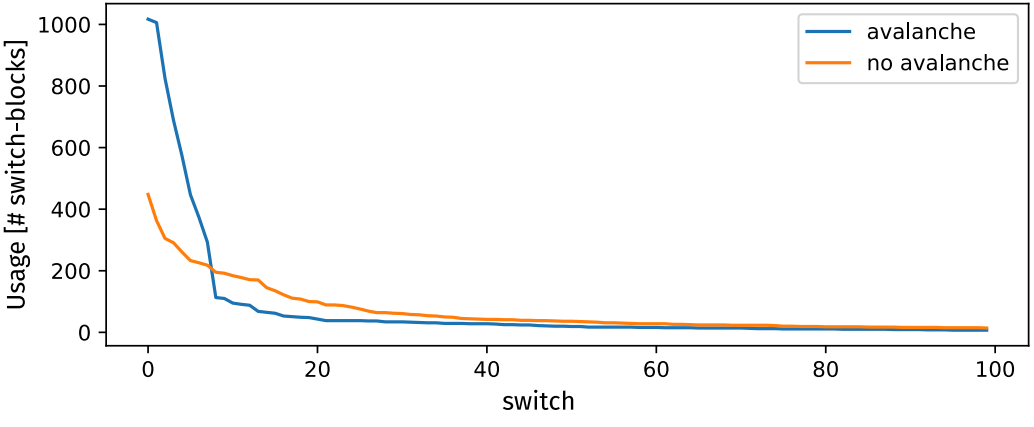
[1] Nikolić and Ienne. Turning PathFinder Upside Down, FPL'21, **Best Paper Award**

Avalanche costs: effect



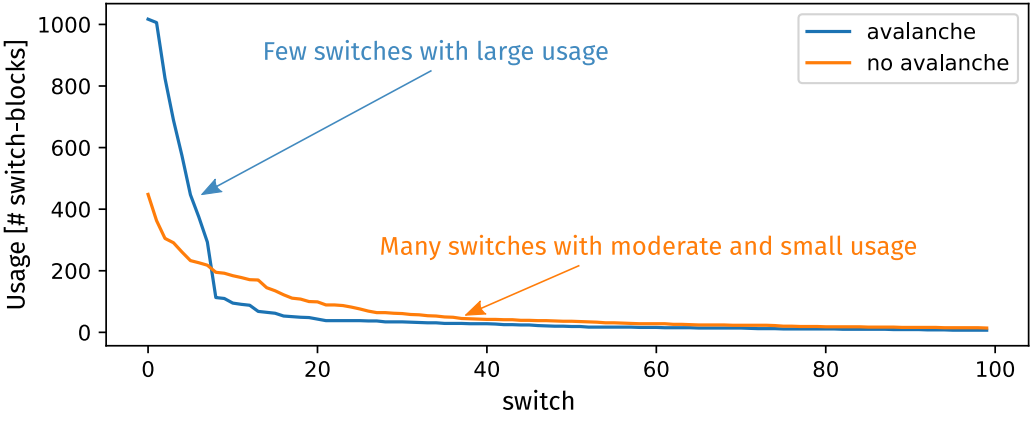
[1] Nikolić and Ienne. Turning PathFinder Upside Down, FPL'21, **Best Paper Award**

Avalanche costs: effect



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

Avalanche costs: effect



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

Avalanche costs: function

Congestion cost: $b(u) \times (1 + p_{fac} \times O(u)) \times (1 + h_{fac} \times C_h(u))$

Avalanche cost: $\max(0, C(u) - (a_p \times U(u) + a_h \times U_h(u)))$

$U(u)$—present number of SBs in which switch u is used

$U_h(u)$—cumulative (historical) number of SBs in which switch u was used

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

Algorithm 1 Simplified PathFinder [McMurchie95, Betz98]

Input: $G = (V, E)$—rr-graph, $E_c \subseteq V \times V$—all connections to be routed; **Output:** A routing tree of each signal

function CONGESTION_COST(u, s) ▷ computes the congestion cost of node u when routing signal s
  if $u \in RT(s)$ **then return** 0 ▷ if u is already used by one connection of s, it can freely be used by another
  return $b(u) \times (1 + p_{fac} \times O(u)) \times (1 + h_{fac} \times C_h(u))$ ▷ otherwise, account for congestion
for $u \in V$ **do**
  $O(u) = 0; C_h(u) = 0$ ▷ set occupancy and historical congestion of all nodes to 0
  if $(\exists v \in V)\ ((u, v) \in E_c)$ **then**
    $RT(u) = \{u\}$ ▷ initialize the routing tree of each signal
$i = 0; p_{fac} = p_{fac}^{init}$
do
  if $i \geq$ max_iter **then return** UNROUTABLE iterations
  for $s \in \{u \in V : (\exists v \in V)\ ((u, v) \in E_c)\}$ **do** iteration;
    this [16]
    for $u \in RT(s)$ **do**
      $O(u) = O(u) - 1$ ▷ reduce the occupancy of all nodes used by the signal s that is ripped up
    $RT(s) = \{s\}; O(s) = O(s) + 1$ ▷ rip up the signal
    for $t \in V : (s, t) \in E_c$ **do**
      $P =$ SHORTEST_PATH$(s, t, \forall u \in V : cong(u) =$ CONGESTION_COST$(u, s))$ ▷ (re)route the connection $s \rightarrow t$
      for $u \in P$ **do**
        if $\neg(u \in RT(s))$ **then**
          $O(u) = O(u) + 1$ ▷ increase the occupancy of all nodes not already used by the signal s
      $RT(s) = RT(s) \cup P$ ▷ add the connection route to the routing tree of s
  for $u \in V$ **do**
    $C_h(u) = C_h(u) + \max(0, O(u) - 1)$ ▷ update historical congestion
  $i = i + 1$
  $p_{fac} = p_{fac} \times p_{fac}^{mult}$ ▷ increase the penalty of using occupied nodes; $p_{fac}^{mult} > 1$ (1.3 is default in VPR [16])
while $\exists u \in V : O(u) > 1$ ▷ finish if there is no congestion
return $\forall RT$ ▷ return all routing trees

Update U(u) as well

Algorithm 1 Simplified PathFinder [McMurchie95, Betz98]

Input: $G = (V, E)$—rr-graph, $E_c \subseteq V \times V$—all connections to be routed; **Output:** A routing tree of each signal

function CONGESTION_COST(u, s) ▷ computes the congestion cost of node u when routing signal s
  if $u \in RT(s)$ **then return** 0 ▷ if u is already used by one connection of s, it can freely be used by another
  return $b(u) \times (1 + p_{fac} \times O(u)) \times (1 + h_{fac} \times C_h(u))$ ▷ otherwise, account for congestion
for $u \in V$ **do**
  $O(u) = 0; C_h(u) = 0$ ▷ set occupancy and historical congestion of all nodes to 0
  if $(\exists v \in V)\ ((u, v) \in E_c)$ **then**
    $RT(u) = \{u\}$ ▷ initialize the routing tree of each signal
$i = 0; p_{fac} = p_{fac}^{init}$
do
  if $i \geq$ max_iter **then return** UNROUTABLE ▷ no congestion-free routing was found in max_iter iterations
  for $s \in \{u \in V : (\exists v \in V)\ ((u, v) \in E_c)\}$ **do** ▷ all signals are ripped up and rerouted in each iteration;
    ▷ modern incremental routers deviate from this [16]
    for $u \in RT(s)$ **do**
      $O(u) = O(u) - 1$ ▷ reduce the occupancy of all nodes used by the signal s that is ripped up
    $RT(s) = \{s\}; O(s) = O(s) + 1$ ▷ rip up the signal
    for $t \in V : (s, t) \in E_c$ **do**
      $P =$ SHORTEST_PATH($s, t, \forall u \in V : cong(u) =$ CONGESTION_COST(u, s)) ▷ (re)route the connection $s \rightarrow t$
      for $u \in P$ **do**
        if $\neg(u \in RT(s))$ **then**
          $O(u) = O(u) + 1$ ▷ y used by the signal s
      $RT(s) = RT(s) \cup P$ ▷ add the connection route to the routing tree of s
  for $u \in V$ **do**
    $C_h(u) = C_h(u) + \max(0, O(u) - 1)$ ▷ update historical congestion
  $i = i + 1$
  $p_{fac} = p_{fac} \times p_{fac}^{mult}$ ▷ increase the penalty of using occupied nodes; $p_{fac}^{mult} > 1$ (1.3 is default in VPR [16])
while $\exists u \in V : O(u) > 1$ ▷ finish if there is no congestion
return $\forall RT$ ▷ return all routing trees

Update $U_h(u)$ as well

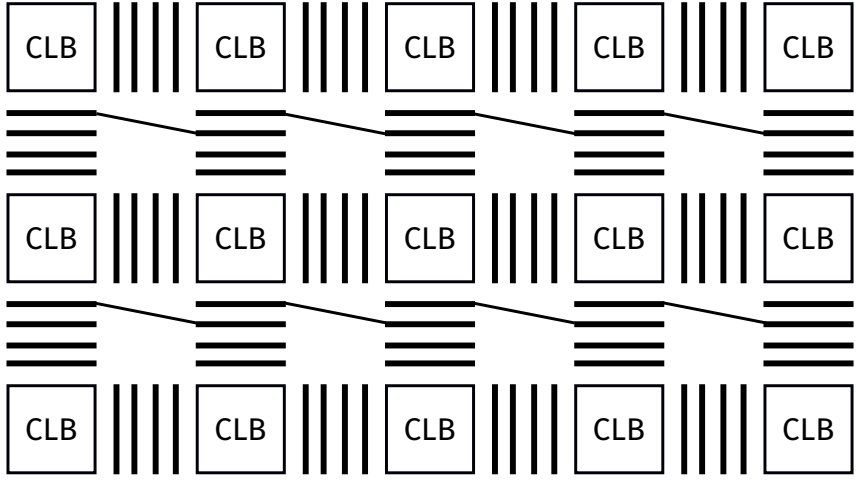
Avalanche vs congestion costs

Congestion

- **UNIQUE** for **EACH** wire instance
- **PENALIZES** using same **WIRE INSTANCES** by different nets

Avalanche

- **COMMON TO ALL** insatnces of the same switch
- **AWARDS** using same **SWITCH TYPES** by different nets



All instances of the same switch share the same avalanche cost

Avalanche vs congestion costs

Congestion

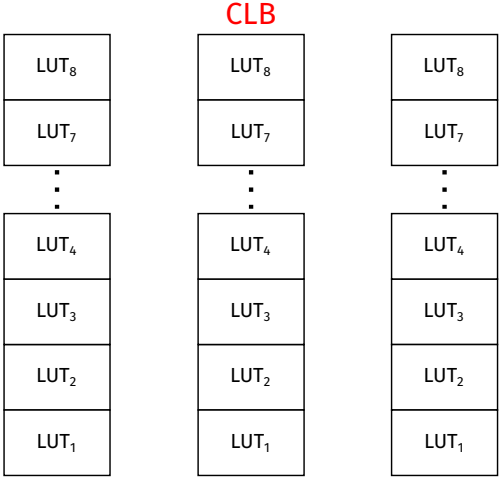
- **UNIQUE** for **EACH** wire instance
- **PENALIZES** using same **WIRE INSTANCES** by different nets

Avalanche

- **COMMON TO ALL** insatnces of the same switch
- **AWARDS** using instances of the same **SWITCH** by different nets

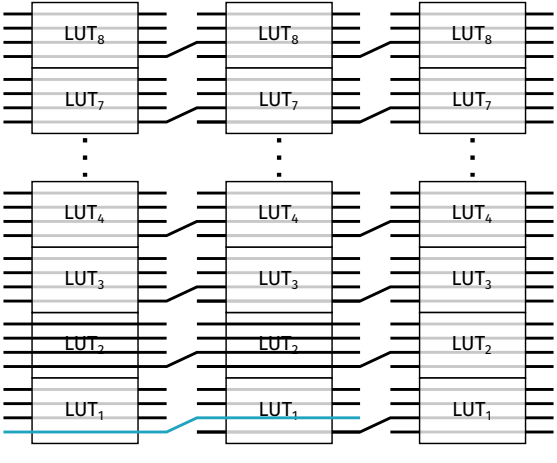
Effectiveness of Avalanche Search

A more physical view of Island-Style FPGAs: stacked LUTs



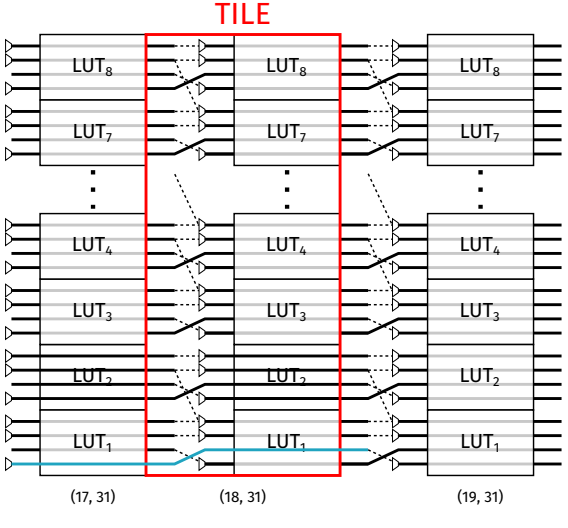
[1] Lewis et al. Architectural Enhancements in Stratix V. FPGA'13

A more physical view of Island-Style FPGAs: wires on top



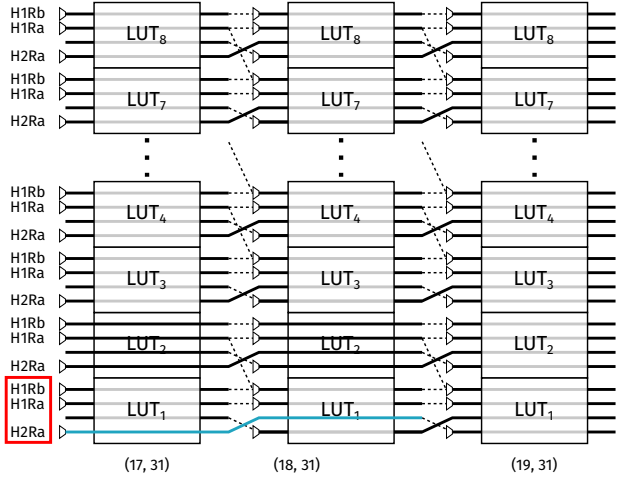
[1] Lewis et al. Architectural Enhancements in Stratix V. FPGA'13

A more physical view of Island-Style FPGAs: muxes on the side



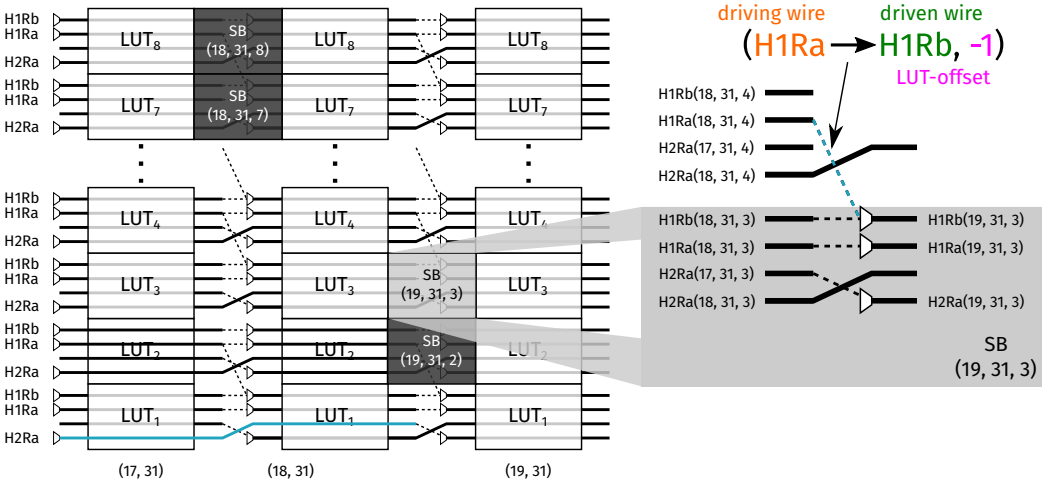
[1] Lewis et al. Architectural Enhancements in Stratix V. FPGA'13

A more physical view of Island-Style FPGAs: distributed channels



[1] Chromczak et al. Architectural Enhancements in Intel Agilex FPGAs. FPGA'20

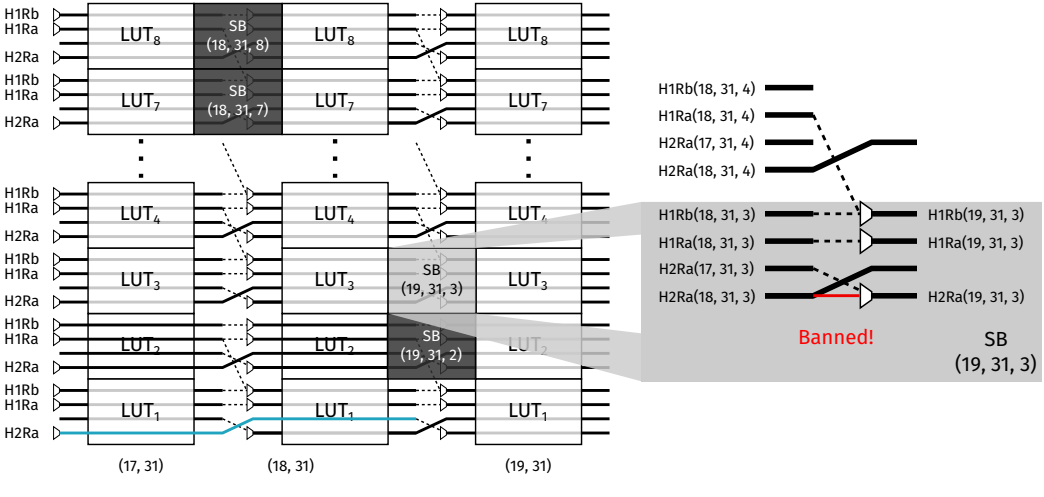
A more physical view of Island-Style FPGAs: distributed SBs



[1] Chromczak et al. Architectural Enhancements in Intel Agilex FPGAs. FPGA'20

(our guess)

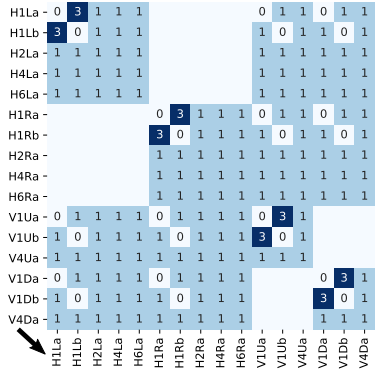
A more physical view of Island-Style FPGAs: no intermediate taps



[1] Chromczak et al. Architectural Enhancements in Intel Agilex FPGAs. FPGA'20

Our best manual effort (4nm; 16 wires; 564 possible switches)

<Fs> = 11.25



#switches	180		
average →	fi	fo	t[ps]
H1	10	10	16.0
H2	11	11	21.3
H4	11	11	30.8
H6	11	11	43.1
V1	12	12	24.6
V4	13	13	74.3
W(tile)	7464 nm		
CPD	1.46 ns		

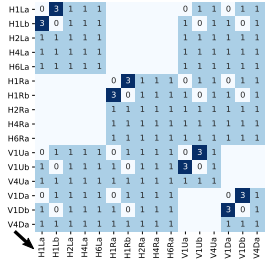
[1] Nikolić, Catthoor, Tőkei, and Ienne. Global Is the New Local. FPGA'21

Avalanche Search

- 3 MCNC circuits used in exploration
- 17 MCNC circuits used for testing
- converged in 36 iterations

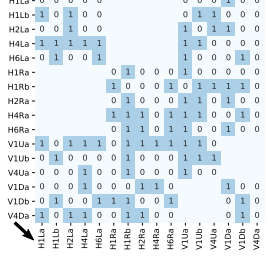
Avalanche Search

<Fs> = 11.25



#switches	180		
average →	fi	fo	t[ps]
H1	10	10	16.0
H2	11	11	21.3
H4	11	11	30.8
H6	11	11	43.1
V1	12	12	24.6
V4	13	13	74.3
W(tile)	7464 nm		
CPD	1.46 ns		

<Fs> = 4.875



#switches	78		
average →	fi	fo	t[ps]
H1	5	3	13.9
H2	5	4	16.8
H4	4	7	27.4
H6	5	5	35.7
V1	7	6	21.8
V4	2	5	70.1
W(tile)	6792 nm -9%		
CPD	1.38 ns -5.48%		

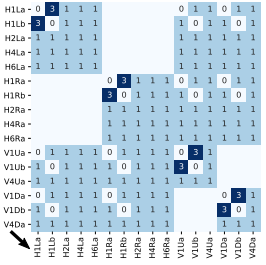
Simulated annealing

- inspired by [1]
- each move is addition or removal of a single switch
- cost function is a combination of tile area and geomean routed critical path delay of the same 3 MCNC circuits
- low-temperature anneal of the manual switch-pattern
- 100 temperature changes
- 100 moves per temperature

[1] Lin, Wawrzynek, El Gamal. Exploring FPGA routing architecture stochastically. TCAD'10

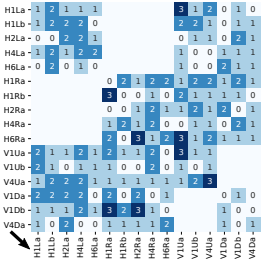
Simulated annealing

<Fs> = 11.25



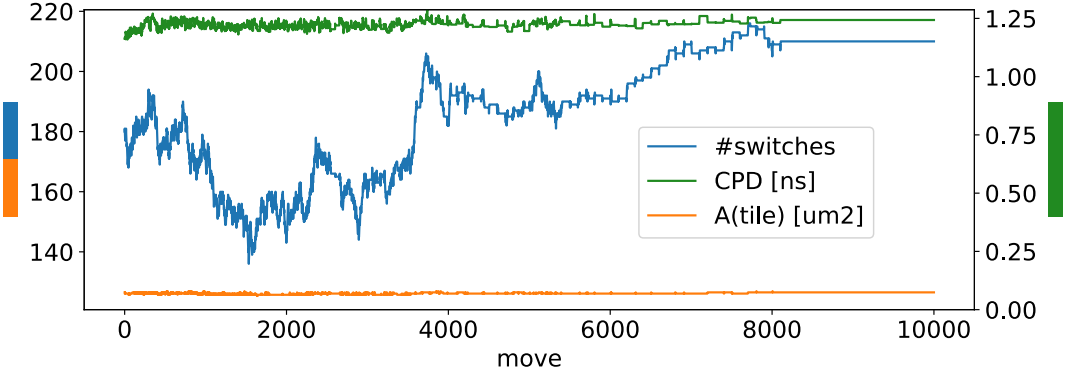
#switches	180		
average →	fi	fo	t[ps]
H1	10	10	16.0
H2	11	11	21.3
H4	11	11	30.8
H6	11	11	43.1
V1	12	12	24.6
V4	13	13	74.3
W(tile)	7464 nm		
CPD	1.46 ns		

<Fs> = 13.125



#switches	210		
average →	fi	fo	t[ps]
H1	13	13	19.6
H2	14	11	24.1
H4	16	12	32.1
H6	9	13	47.3
V1	14	15	29.2
V4	13	15	86.8
W(tile)	7488 nm +0.32%		
CPD	1.55 ns +6.16%		

Simulated annealing



With a better cost function, directed moves, etc., it would likely work better

Routability

Testing on

- 10 Gnl [1] circuits
- each with ~ 10 000 LUTs
- all with Rent's exponent 0.7

[3] Stroobandt, Depreitere, and Campenhout. Generating new benchmark designs. Integration'99

Routability

Congested nodes after 300 PathFinder iterations [%]

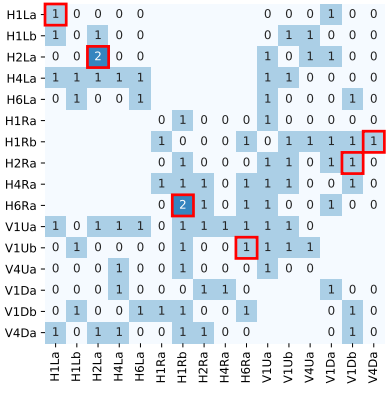
circuit	1	2	3	4	5	6	7	8	9	10
avalanche	0.906%	0.274%	0%	0.521%	0.323%	0.187%	0.149%	0.262%	0.040%	0.002%

- only 1/10 circuits was routed
- sparsification of the switch-pattern was enabled by MCNC circuits not requiring the channel width dimensioned for larger circuits

Routability: remedy

Perform additional iterations of Avalanche Search,
this time using one Gnl circuit per iteration

Routability: remedy



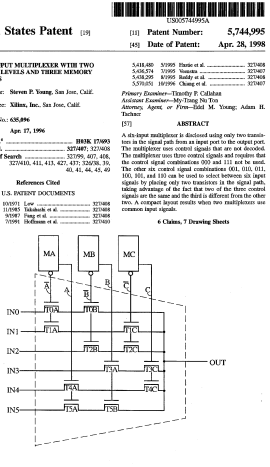
Number of iterations until congestion is resolved

circuit	1	2	3	4	5	6	7	8	9	10
Gnl-extended avalanche	142	61	27	106	26	55	46	55	30	82

- converged in 3 iterations
- added 6 more switches (now 84)
- routed delay went up from 1.38 to 1.39 ns
- all 10 Gnl circuits are routable

Regularizing Avalanche Search

Layout prefers regularity



US005744995A

United States Patent [19]
Young

[11] **Patent Number: 5,744,995**
[45] **Date of Patent: Apr. 28, 1998**

[54] **SIX-INPUT MULTIPLEXER WITH TWO GATE LEVELS AND THREE MEMORY CELLS**

[75] Inventor: **Steven P. Young**, San Jose, Calif.

[73] Assignee: **Xilinx, Inc.**, San Jose, Calif.

[21] Appl. No.: **635,096**

[22] Filed: **Apr. 17, 1996**

[51] **Int. Cl.**[6] **H03K 17/693**
[52] **U.S. Cl.** **327/407**; 327/408
[58] **Field of Search** 327/99, 407, 408, 327/410, 411, 413, 427, 437; 326/38, 39, 40, 41, 44, 45, 49

References Cited

U.S. PATENT DOCUMENTS

3,614,327	10/1971	Lew	327/408
4,551,634	11/1985	Takahashi et al.	327/408
4,692,634	9/1987	Pang et al.	327/408
5,698,641	7/1991	Hoffman et al.	327/403
5,418,480	5/1995	Florin et al.	327/408
5,436,574	7/1995	Veenstra	327/407
5,518,295	8/1995	Reddy et al.	327/408
5,570,050	10/1996	Chang et al.	327/407

Primary Examiner—Timothy P. Callahan
Assistant Examiner—My-Trang Nu Ton
Attorney, Agent, or Firm—Edel M. Young; Adam H. Tachner

[57] **ABSTRACT**

A six-input multiplexer is disclosed using only two transistors in the signal path from an input port to the output port. The multiplexer uses control signals that are not decoded. The multiplexer uses three control signals and requires that the control signal combinations 000 and 111 not be used. The other six control signal combinations 001, 010, 011, 100, 101, and 110 can be used to select between six input signals by placing only two transistors in the signal path, taking advantage of the fact that two of the three control signals are the same and the third is different from the other two. A compact layout results when two multiplexers use common input signals.

6 Claims, 7 Drawing Sheets

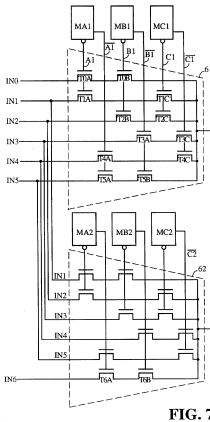
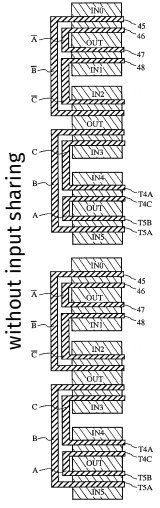
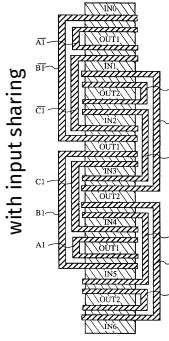


FIG. 7





Layout prefers regularity

If multiplexers are 6:1:

- fewer memory cells are needed (normal 2-level 6:1 needs 4 SRAMs)

If two multiplexers share 5/6 inputs:

- much more compact layout (diffusion sharing)
- fewer vias
- shorter wiring

Layout prefers regularity

Can we constrain Avalanche Search to produce only regular solutions?

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
4. if no unmarked switches are used, done
5. mark n most-used unmarked switches and set their cost to 0
6. goto 3

Key points: constructing a feasible solution

After step 3 (PathFinder routing):

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. Mark *n* most-used unmarked switches and reduce their cost to 0

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

1. Assign full cost with no discount

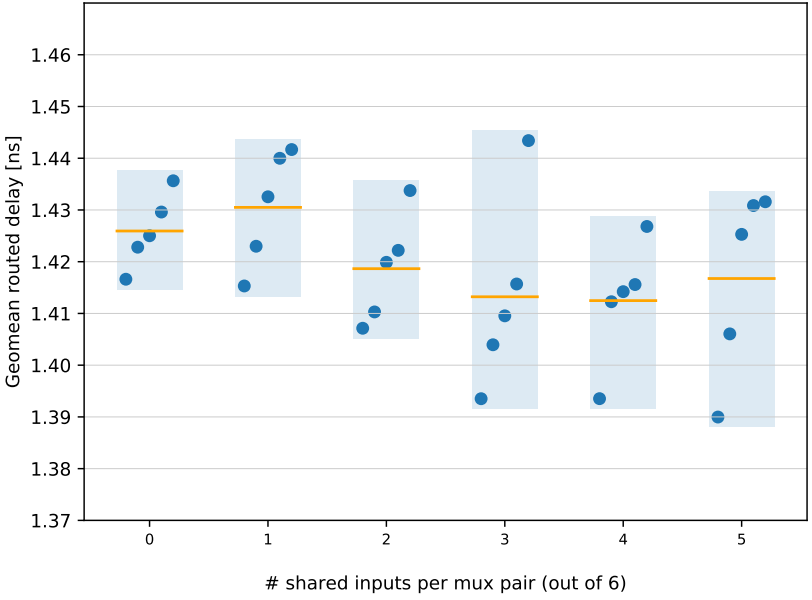
Key points: ensuring that constraints are met

- Final pattern is the last ILP solution

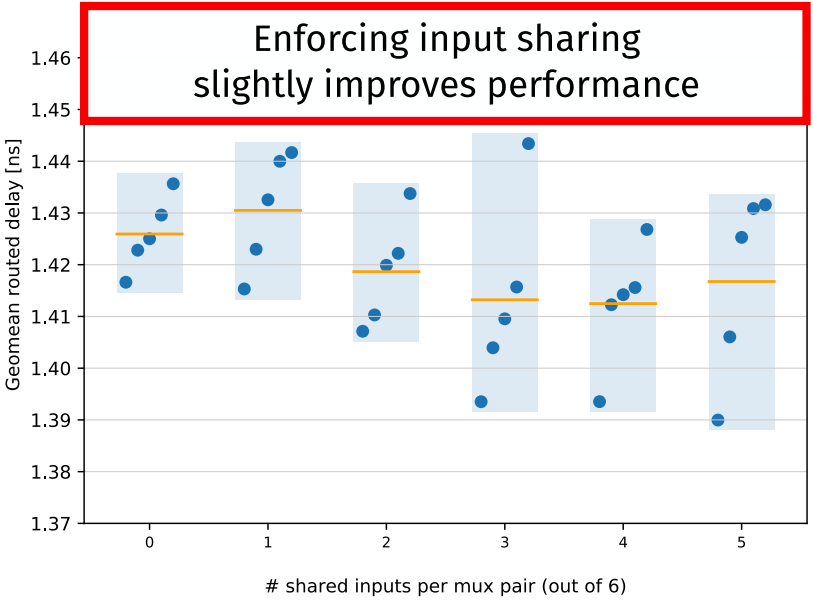
Regularizing Avalanche Search

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

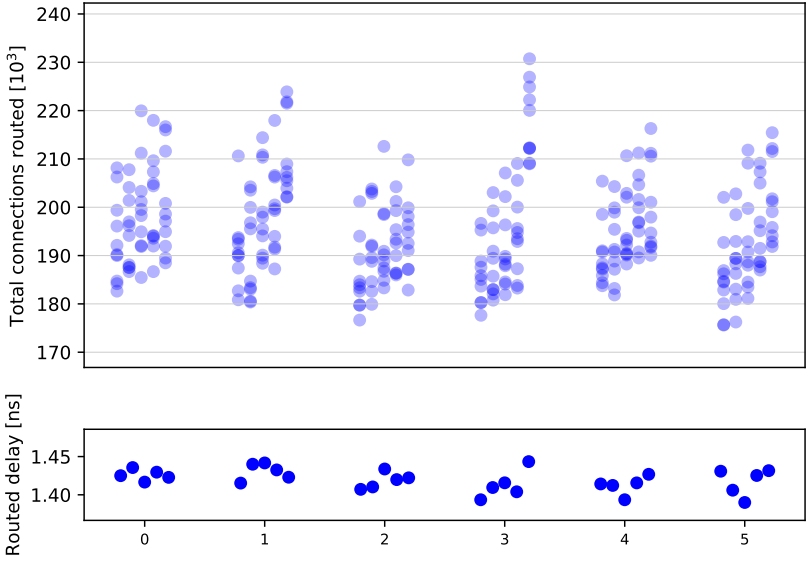
Input sharing: delay



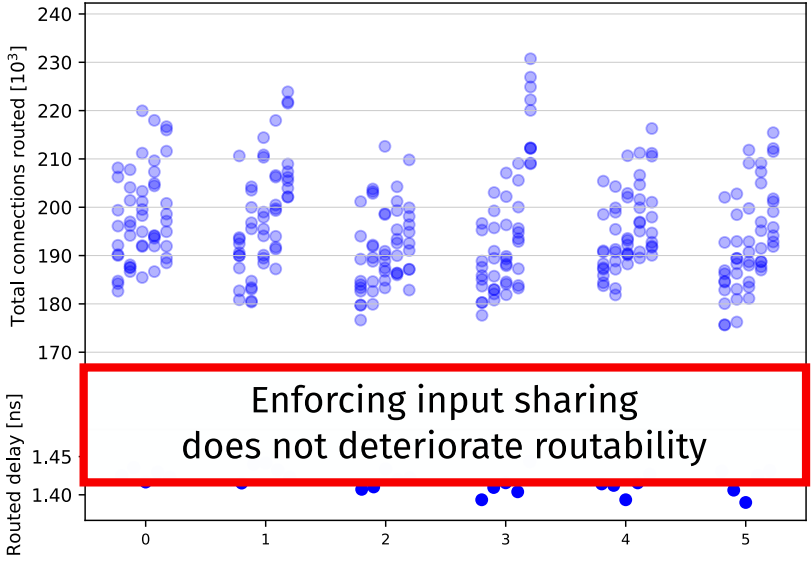
Input sharing: delay



Input sharing: routability



Input sharing: routability



Runtime scalability

Routing graph size

- embedding the entire design space in the routing graph increases its size significantly
- this slows down shortest path search

A*

- avalanche costs can drop to zero
- for the lookahead to be admissible, we must compute it for all avalanche costs at zero
- this makes it very ineffective when switch usage is low (costs $\sim 1000\times$ base costs)

A*: possible remedy

- store the minimum number of unmarked nodes to reach each tile
- multiply this number by the current lowest cost of an unmarked switch
- maybe some more preprocessing can be afforded as avalanche costs are common to all instances of the same switch, unlike congestion

Conclusions

- we now have a method for automatically designing switch-patterns without explicitly listing and testing solutions (previously a fundamental bottleneck)
- we can also make the solutions respect arbitrary constraints
- because the method operates on a routing graph, it could be useful for other aspects of interconnect (e.g., intracluster connectivity, channel segmentation, etc.)
- more work is needed to resolve the A* issues