Regularity Matters: Designing Practical FPGA Switch-Blocks

Stefan Nikolić
École Polytechnique Fédérale de Lausanne (EPFL)
Lausanne, Switzerland
stefan.nikolic@epfl.ch

Paolo Ienne
École Polytechnique Fédérale de Lausanne (EPFL)
Lausanne, Switzerland
paolo.ienne@epfl.ch

ABSTRACT

Several techniques have been proposed for automatically searching for FPGA switch-blocks which typically show some tangible advantage over the well-known academic architectures. However, the resulting switch-blocks usually exhibit high levels of irregularity, in contrast with what can be observed in a typical commercial architecture. One wonders whether the architectures produced by such search methods can actually be laid out in an efficient manner while retaining the perceived gains. In this work, we propose a new switch-block exploration method that enhances a recently published search algorithm by combining it with ILP, in order to guarantee that the obtained solution satisfies arbitrary regularity constraints. We measure the impact of regularity constraints commonly seen in industrial architectures (such as limiting the number of different multiplexer sizes or forced sharing of inputs between pairs of multiplexers) and observe benefits to the routability of complex circuits with only a limited reduction in performance.

CCS CONCEPTS

• **Hardware → Programmable interconnect**; **Methodologies for EDA**; **Wire routing**; *Design for manufacturability*.

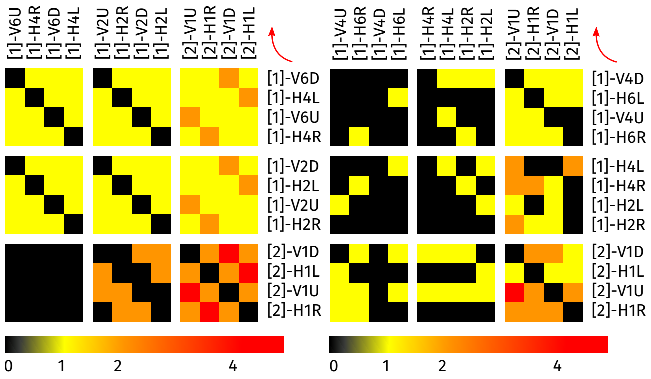
KEYWORDS

interconnect, routability, automated exploration, layout, regularity, multiplexer, input sharing, constraint, ILP, optimization, avalanche

ACM Reference Format:
Stefan Nikolić and Paolo Ienne. 2023. Regularity Matters: Designing Practical FPGA Switch-Blocks. In *Proceedings of the 2023 ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '23), February 12–14, 2023, Monterey, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3543622.3573192

1 WHO CARES ABOUT "REGULARITY"?

Figure 1 shows wire adjacency of two switch-blocks: one from a 7-Series FPGA [1] (Figure 1a) and the other a product of automated exploration attempting to optimize the switch-block for a set of relevant circuits [2] (Figure 1b). Although the matrices are not directly comparable as their wire sets differ, one thing immediately draws attention: the commercial switch-block is perfectly "regular", while the one resulting from automated exploration is very "irregular" in comparison. Could it be that imposing regularity is simply too constraining for the switch-block to reach peak performance? Or could there perhaps be other solutions with comparable performance that are also regular? Are there any downsides of irregularity, apart from the evident increase in the already high layout effort needed to produce an FPGA? If significantly different circuits are implemented on the optimized architecture, will it still outperform the regular commercial one, or will it become unroutable?



(a) Commercial switch-block. **(b) Result of automated search.**

Figure 1: Regularity of a commercial switch-block contrasted with an irregular solution obtained through automated search. Figure (a) shows simplified wire adjacency in a 7-Series Xilinx FPGA [1]. An analogous plot of a switch-block obtained by automated search [2] is shown in Figure (b). Although wire sets differ between the two switch-blocks and they can thus not be directly compared, a stark difference in "regularity" is readily observed.

All these are very interesting questions that, to the best of our knowledge, have not been systematically answered until now. In fact, most published search methods would produce highly irregular solutions (see Section 2) without much regard to whether they can be fabricated while retaining the observed performance gains. In this work, we attempt to correct this. Apart from providing insight into benefits and downsides of various forms of regularity that commercial switch-blocks have been observed to possess, we propose a general algorithm for searching for switch-blocks with arbitrary forms of regularity. This includes those which may arise in the future, due to technology and/or CAD algorithm evolution.

The proposed algorithm is introduced in Section 4. It consists of integrating an *Integer Linear Programming* (ILP)-based switch-block construction method with our recent *Avalanche Search* method [2] which generates switch-blocks by greedily taking switches that a negotiation-based router displays the most affinity to. A review of related work is given in Section 2 and Avalanche Search in particular in Section 3. Subsequent sections present the use of the developed algorithm for a thorough investigation of the impact of various types of switch-block regularity collected primarily from the available data about real commercial architectures, but also prior

FPGA '23, February 12–14, 2023, Monterey, CA, USA.

ACM ISBN 978-1-4503-9417-8/23/02.
https://doi.org/10.1145/3543622.3573192

academic work. Concerns about ILP solution time are addressed in Section 12, while conclusions are drawn in Section 13.

2 RELATED WORK

Several methods for automating switch-block exploration have been proposed [2–4], with simulated annealing forming the basis for majority of them. Due to randomized search and/or noise generated by the CAD tools used in the exploration loop, the architectures they produce tend to be highly irregular.

Nevertheless, such techniques that enable quick architectural optimization can be highly useful, even if the solutions they produce may not be easily fabricated in an actual product. A notable example of this is found in the work of Lemieux et al., which clearly demonstrated that sparse crossbars can bring large area reductions without sacrificing routability [5]. While the key idea of sparsifying the crossbar was carried over to Stratix I, for different layout reasons, the actual implemented pattern was considerably more regular [6]. This also enabled simple LUT output swapping by the router without having to reconsider the local routing of the input signals: an interesting example of the case when constraints imposed on the interconnect architecture lead both to an efficient layout and a possibility to improve CAD efficiency.

When an automated search method is not intended to test a groundbreaking hypothesis, such as the one in the work of Lemieux et al., but to further optimize architectures from a known design space, inability to easily fabricate the solutions may question the actual utility of the method. Unfortunately, all of the papers cited at the beginning of this section propose methods that fall in this second category. In this work, we build on top of the most recent of them—Avalanche Search [2]—to produce practical switch-blocks.

3 REVIEW OF AVALANCHE SEARCH

The goal of the Avalanche Search algorithm [2] is to select a subset SOL of manufacturable switch types E, which connect wire types from a fixed set V, such that SOL is as small as possible but still allows the router to route all circuits of interest, while meeting timing constraints. The switch types entering SOL constitute a *switch-pattern* which is repeated in every FPGA tile, next to every LUT [7]. Each switch type from E is designated by an ordered triplet (u, v, d_L), where $u \in V$ is the driving wire type, $v \in V$ the driven wire type, and d_L the distance between the end of the driving wire and the start of the driven wire in terms of the number of LUTs (see Fig. 1 of Nikolić and Ienne [2]).

The algorithm is outlined in Figure 2. It iteratively builds a switch-pattern by 1) routing circuits and observing in how many switch-blocks each switch type was used (*usage*, $U((u, v, d_L))$; box ②) and 2) increasing the pattern by a certain number of most-used switch types (box ⑤), until there are no more switch types used by the router which are not already in the pattern (boxes ③ and ④). To be able to measure usage of switch types that are not yet part of the pattern, each switch type that could be fabricated is represented in the routing-resource graph of box ②. These switch types, however, are represented with a high initial cost which drops in proportion to usage, whereas the switch types that are part of the pattern are assigned zero cost and their usage is no longer measured. As a result, the set of all switch types used by the router (*tentative solution*, $tSOL$, in box ③) slowly converges to the increasing set of switch types accepted into the pattern (*partial solution*, $pSOL$).

4 REGULARIZATION ALGORITHM

The algorithm of Figure 2 provides no mechanism for controlling the structure of the solution. We will now extend it to impose arbitrary constraints, while retaining tight coupling with the router.

4.1 General Flow

The proposed algorithm is shown in Figure 3. It retains the structure of the algorithm of Figure 2, with two important modifications. First, the tentative solution, $tSOL$, is not formed by merely observing which switch types were used by the router. An ILP problem is constructed instead, such that all of its feasible solutions satisfy all specified regularity constraints. Maximizing affinity of the router towards the switch types that enter $tSOL$, represented by their usage, increases the likelihood that the final solution is routable and appropriate for critical path optimization of the routed circuits.

Second, selection of the most-used switch types from $tSOL$ which enter the final pattern in box ⑤ is identical to that of Figure 2, but the initial avalanche cost [2] of the remaining switch types in $tSOL$ is set to a lower value in the next iteration of the algorithm. The rationale is as follows: in the first iteration, the router uses switches unaware of the constraints imposed on the switch-pattern. This may result in the set of used switch types being very far from meeting the constraints. The ILP solver legalizes the solution, respecting the decisions of the router as much as possible by maximizing the total usage. In the second iteration, those switch types which are known to be part of at least one legal solution ($tSOL$) are offered to the router at a reduced price, making it less likely for the router to violate the constraints again. Over time, the router and the ILP solver converge towards a common solution.

In order to guarantee that the algorithm ends, once a switch type enters the partial solution, $pSOL$, it is never removed from it; this is the same as in Figure 2. Moreover, since $pSOL$ is always extended only by the switch types from $tSOL$, produced by the ILP solver and hence legal by construction, the algorithm is always guaranteed to end with a solution that satisfies all of the imposed constraints. It could happen, however, that the final solution does not encompass all of the switch types that the router used in the last iteration, because adding them to $tSOL$ would make it violate the constraints. These switch types may either slightly improve the delay of some (near-)critical paths or be necessary for complete removal of congestion. The second case would result in the constraint set being deemed *unsatisfiable* and it would have to be relaxed. We note, however, that in all experiments performed in preparation of this paper, we never encountered such a situation.

4.2 Base ILP Problem

In this section, we describe a skeleton ILP problem which completes the algorithm of Figure 3. Different types of regularity constraints will be gradually added to it over the subsequent sections.

Given a switch type e connecting a wire type u to a wire type v at a LUT distance d_L [2], we designate its presence in the switch-pattern by the following binary variable:

$$x_{u,v,d_L} \in \{0, 1\}, \qquad \forall (u, v, d_L) \in E. \tag{1}$$

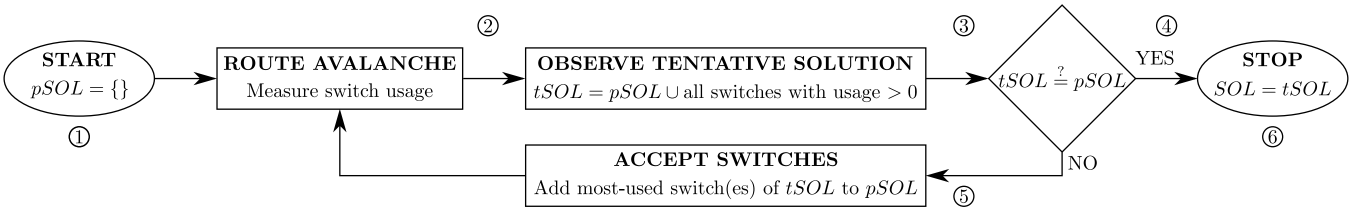

START
$pSOL = \{\}$
①
②
ROUTE AVALANCHE
Measure switch usage
③
OBSERVE TENTATIVE SOLUTION
$tSOL = pSOL \cup$ all switches with usage > 0
$tSOL \overset{?}{=} pSOL$
④
YES
STOP
$SOL = tSOL$
⑥
NO
ACCEPT SWITCHES
Add most-used switch(es) of $tSOL$ to $pSOL$
⑤

Figure 2: Avalanche Search algorithm [2].

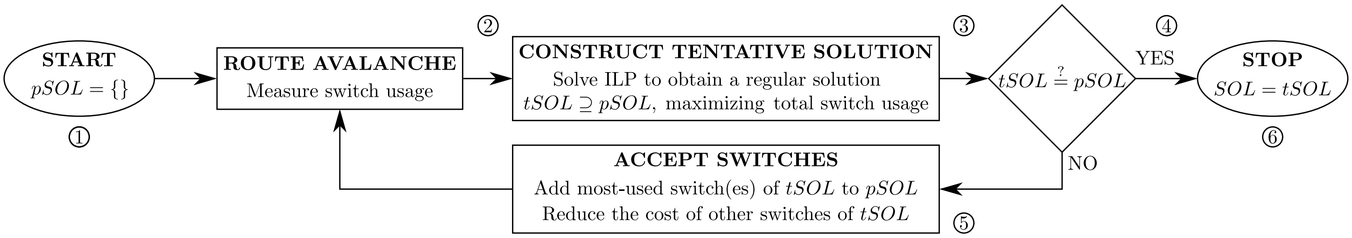

START
$pSOL = \{\}$
①
②
ROUTE AVALANCHE
Measure switch usage
③
CONSTRUCT TENTATIVE SOLUTION
Solve ILP to obtain a regular solution
$tSOL \supseteq pSOL$, maximizing total switch usage
$tSOL \overset{?}{=} pSOL$
④
YES
STOP
$SOL = tSOL$
⑥
NO
ACCEPT SWITCHES
Add most-used switch(es) of $tSOL$ to $pSOL$
Reduce the cost of other switches of $tSOL$
⑤

Figure 3: Search algorithm proposed in this work, which enforces arbitrary regularity constraints on the obtained solution.

The corresponding switch type is part of the switch-pattern iff the variable is 1. To specify that the accepted switch types must be part of $tSOL$, we simply set the appropriate presence variables to 1:

$$x_{u,v,d_L} = 1, \qquad \forall (u, v, d_L) \in pSOL. \tag{2}$$

As mentioned in Section 4.1, the basic objective function strives to maximize the total usage of the switch types entering the solution:

$$\max \sum_{(u,v,d_L) \in E} U((u, v, d_L)) x_{u,v,d_L}. \tag{3}$$

Usage of each switch type is observed from the router and is thus a constant in the ILP problem. To prevent selection of all switch types in absence of further constraints, each ILP imposes an upper bound M on switch-pattern size:

$$\sum_{(u,v,d_L) \in E} x_{u,v,d_L} \leq M. \tag{4}$$

5 EXPERIMENTAL SETUP

All of the subsequent sections that progressively introduce different kinds of regularity constraints share a common experimental setup adopted from our previous work [2]. Most notably, all experiments are performed on an eight 6-LUT-based architecture with a fixed channel segmentation containing wires indicated in Figure 1b, where *R, L, U,* and *D* designate wires going right, left, up, and down, respectively. This segmentation was inspired by that of Agilex [8], with omission of long wires. It gives an equivalent horizontal channel width of $8 \times 2 \times (1+1+2+4+6) = 224$ and an equivalent vertical channel width of $8 \times 2 \times (1 + 1 + 4) = 96$. Switches between wires in the same plane (starting and ending in the vicinity of one LUT of the CLB [2, 8]) as well as between adjacent planes are allowed, while U-turns are prohibited. This leads to 564 switch-types in total.

The only additional parameter introduced in the algorithm of Section 4 is the cost reduction factor for switch types participating in the ILP solution ($tSOL$) that have not yet been accepted into the partial solution ($pSOL$). We fix this to 0.9, which was experimentally determined to yield slightly better results than other tested values.

In order to suppress the effects of experimental noise that could lead to false conclusions about which trade-offs a certain set of constraints brings, we need to compare sets of feasible architectures, rather than single points. To this end, we leverage the well-known fact that permuting the order in which the nets of a circuit are routed can have a significant impact on the outcome of the routing process [9, 10] and hence also on the usage of different switch types. Permutation is achieved by performing 100 random swaps in the default-sorted netlist [11]. For each constraint set, we construct five different architectures, by permuting the netlist using five different random number generator seeds.

The switch-pattern size is bounded by 96 in all experiments, which allows finding solutions for all constraint types introduced in the subsequent sections, yet, does not make the patterns excessively larger than the ones produced by Avalanche Search [2].

6 LIMITING MULTIPLEXER SIZE VARIATION

The first form of regularity apparent in the industrial architectures is the very limited number of multiplexer sizes in their switch-patterns—often only one [1, 12, 13]. The reason for such uniformity likely lies in the efficiency with which multiplexers of certain sizes can be laid out [1, 14], but there could also be other reasons such as increased routability. In this section, we investigate the benefits and downsides of limiting the number of multiplexer sizes. When the size of the multiplexer driving each wire v is known in advance to be some constant $\varphi(v)$, extending the ILP problem of Section 4.2 to respect this size distribution is trivial:

$$\sum_{u,d_L:(u,v,d_L) \in E} x_{u,v,d_L} = \varphi(v), \qquad \forall v \in V. \tag{5}$$

However, we would like to measure the impact of limiting the number of multiplexer sizes in general, without a priori assigning a size to any particular multiplexer, since this unrelated decision could influence the conclusions.

6.1 Encoding

Given a maximum allowed size for any multiplexer, M_φ, for each size $\varphi \in [0, M_\varphi]$, and each wire v, we introduce another binary variable $x_{v,\varphi}$, which is 1 iff the size of the fanin of the wire v is φ:

$$\sum_{u,d_L:(u,v,d_L)\in E} x_{u,v,d_L} = \sum_{\varphi\in[0,M_\varphi]} \varphi x_{v,\varphi}, \qquad \forall v \in V, \tag{6}$$

$$\sum_{\varphi\in[0,M_\varphi]} x_{v,\varphi} = 1, \qquad \forall v \in V. \tag{7}$$

For each allowed multiplexer size φ, we introduce another binary variable, x_φ, indicating that there is at least one wire in the switch-pattern driven by a multiplexer of that size:

$$x_\varphi = \bigvee_{v\in V} x_{v,\varphi}, \qquad \forall \varphi \in [0, M_\varphi]. \tag{8}$$

For each φ, the above disjunction is linearized in a standard way [15]:

$$\sum_{v\in V} x_{v,\varphi} \geq x_\varphi, \tag{9}$$

$$x_{v,\varphi} \leq x_\varphi, \qquad \forall v \in V. \tag{10}$$

Finally, we need to limit the number of different multiplexer sizes present in the solution to the desired constant N_φ:

$$\sum_{\varphi\in[0,M_\varphi]} x_\varphi \leq N_\varphi. \tag{11}$$

If distribution of a specific set of sizes, Φ (e.g., $\Phi = \{6, 20, 25\}$ [1]) is sought, $[0, M_\varphi]$ is simply replaced by Φ in the above equations.

6.2 Results

We investigate the difference in performance between architectures with up to one, two, four, or any number of arbitrary different multiplexer sizes up to 20:1 ($N_\varphi \in \{1, 2, 4, \infty\}$, $M_\varphi = 20$).

Critical Path Delay: Figure 4 shows the critical path delays when routing MCNC circuits on the obtained architectures. Each point is a geometric mean of 17 circuits and each circuit is in turn represented by the median over five different placements [2]. Although there are significant differences between architectures within each group, we can observe that allowing two rather than a single multiplexer size provides slightly more stable results, while increasing the number to four leads to a clearer advantage. Architectures obtained when the number of multiplexer sizes is not bounded clearly outperform the others, with a close to 3% average critical path delay reduction over the case when only one size is allowed. This demonstrates that there is a performance advantage to be reaped when it is possible to tailor the multiplexer sizes specifically to the needs of the circuits of interest.

Architectures labeled as "∞, 96 sw." also do not limit the number of multiplexer sizes, but instead of considering $M = 96$ an upper bound on switch-pattern size, they all have exactly 96 switch types. In comparison, the ∞-labeled architectures have 72–86 switch types. As a result, capacitive loading on wires is increased and the advantage of freely choosing multiplexer sizes is largely lost.

Routability: Only three circuits were used in the search [2], yet, all 17 circuits on which performance comparisons are based were routable. Nevertheless, the remaining 14 circuits come from the same benchmark set, so it is of interest to see how routable the patterns are when routing circuits from a more complex set. To this end, we attempt routing ten 10 000 LUT circuits with a Rent's exponent of 0.7, generated by Gnl [2, 16]. Results of these experiments are shown in Figure 5. Since modern routers, including VTR-8, are incremental [11], we plot the total number of routed connections taken to complete the routing of each circuit on each architecture, as we believe that this is a more telling metric of the difficulty which the router faces than the more common iteration count. Geometric means of the iteration count and the wirelength of all circuits,

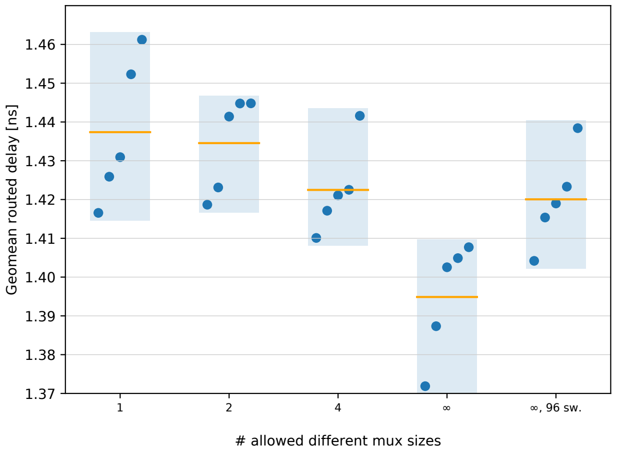


Figure 4: Geomean critical path delays after routing the MCNC benchmarks. Each point represents a single architecture. Average delay among the architectures within the same constraint set is depicted by an orange horizontal line. Blue rectangles are used merely for visual separation of architectures obtained for different constraint sets. Unconstrained architectures (∞) clearly have a performance advantage.

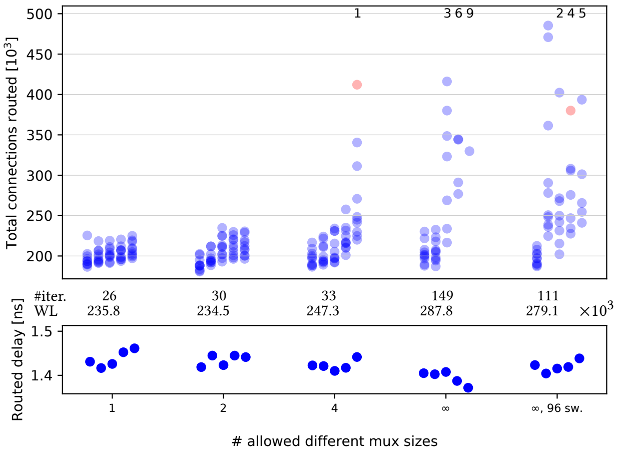


Figure 5: Routability on Gnl circuits. Each point represents the total number of connections that were routed (including rip-ups) when routing a single Gnl circuit on a particular architecture. Circuits that failed to route are shown in red. Since for many of the unroutable circuits more connections were rerouted than can fit in the plot, the number of failed circuits for each architecture is written at the top. All numbers are single-digit (there were 10 circuits in total). The plot below shows the average critical path delay on the MCNC circuits for the particular architecture as a reference (see plot of Figure 4). While unconstrained architectures were better optimized to achieve good performance on circuits from the set used to construct them, they struggle to route significantly more complex circuits. Multiplexer size regularity has a clear advantage in routability of more complex circuits.

Table 1: Generalization.

N_φ	delay [ns]	# iter.	# routed con. [10^3]	WL [10^3]
		exploration circuits		
1	1.143	29	15.82	6.35
2	1.147	29	15.97	6.08
4	1.133	29	14.68	5.86
∞	1.113	29	16.72	6.01
∞, 96 sw.	1.131	29	16.22	6.00
		other MCNC circuits		
1	1.510	30	63.02	22.53
2	1.505	30	63.87	21.42
4	1.493	31	61.16	20.83
∞	1.464	31	66.50	21.48
∞, 96 sw.	1.491	31	68.10	21.50
		Gnl circuits		
1		26	202.14	235.84
2		30	204.37	234.46
4		42	219.86	247.25
∞		149	581.91	287.80
∞, 96 sw.		111	343.75	279.13

averaged over all five architectures of the given constraint set are shown in rows labeled as "#iter." and "WL", respectively.

There is little difference in the number of connections that have to be routed between the architectures which have one or two multiplexer sizes. However, as soon as N_φ increases to four, there is a very clear drop in routability, with one architecture even failing to route all circuits. This trend further increases when the number of multiplexer sizes is unbounded, regardless of the switch-pattern size constraint: in both cases, three of the five unbounded architectures fail to complete some of the circuits within 300 iterations.

Table 1 shows the average routability metrics of architectures obtained for different constraint sets on circuits split into three groups: 1) those used in exploration, 2) the remaining circuits from the MCNC set, and 3) the more complex circuits from the Gnl set. Failed circuits enter the geometric average for a given architecture with 300 iterations, which was the limit used in the experiments. We can observe that the highly delay-optimized architectures with no constraint on multiplexer size count are somewhat less routable (albeit with a competitive wirelength) even on the circuits used for exploration. This trend is maintained on the remaining MCNC circuits, while the advantage of the regular architectures becomes really significant on the larger Gnl circuits. It seems quite clear that having uniform multiplexer sizes makes it much more likely that the architecture will support circuits with characteristics not captured by those used during the search. This is certainly an important aspect for FPGA vendors, and it is likely better to give up some of the potential performance benefits that a less regular architecture could bring, in order to make it significantly more general.

7 LIMITING FANOUT SIZE VARIATION

In the previous section, the number of different multiplexer sizes that occur in the switch-pattern was bounded, but there were no constraints on the fanout size variation. However, large variations in fanout size may make it more difficult to optimize the critical path delay, due to differences in capacitive loads on wires, unless the router is able to effectively exploit them [8]. In this section, we assess the effect of constraining fanout and multiplexer size count together. Constraints that are needed for this are dual to the ones of Section 6.1, with edges being listed with respect to the tail node.

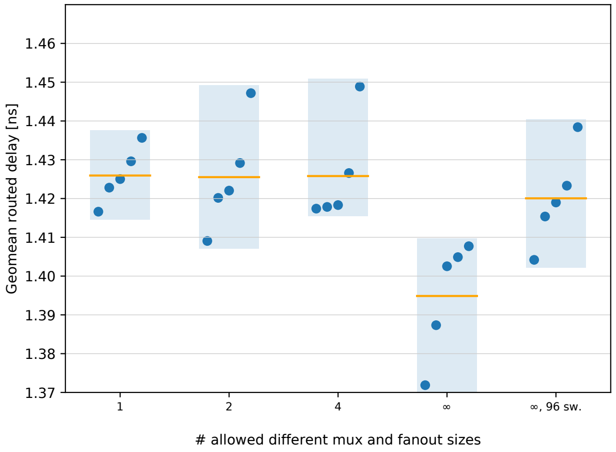


Figure 6: Influence of bounding the number of different multiplexer and fanout sizes on the critical path delay of the implemented MCNC circuits. Regularizing fanout sizes balances capacitive loads and reduces the negative impact of multiplexer size regularization on delay.

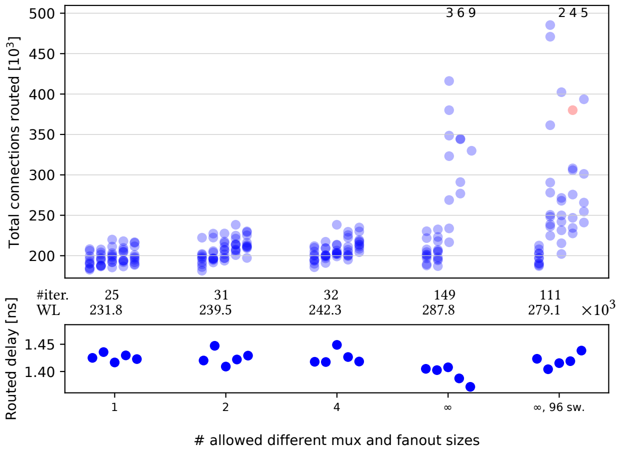


Figure 7: Influence of bounding the number of different multiplexer and fanout sizes on routability. Regularizing fanout sizes further improves it.

7.1 Results

All experiments are set up exactly as in Section 6. We bound the number of multiplexer and fanout sizes by the same constant.

Critical Path Delay: Influence of regularizing fanout sizes on critical path delay of the routed circuits is shown in Figure 6. More balanced capacitive loads and more options that the router has to avoid congestion bottlenecks leads to improved results when the number of multiplexer sizes is very limited ($N_\varphi \in \{1, 2\}$). However, this improvement is still not sufficient to make the regularized architectures outperform the unconstrained ones, where the router can freely select the switch types that form the pattern.

Routability: Influence that regularizing fanout sizes has on routability is shown in Figure 7. When compared with Figure 5, we can see that regularization of fanout sizes contributes to additional improvement of routability. This is the most apparent when the number of sizes is bounded by four, where now all architectures manage to route all circuits, but it also brings improvement to the architectures with completely uniform fanins. Balancing the fanout

Table 2: Average total wire delay with respect to the number of inputs shared.

# inputs shared	0	1	2	3	4	5
t [ps]	223.2	221.6	220.6	220.4	220.4	218.6
Δ [%]	0.00	−0.72	−1.16	−1.25	−1.25	−2.06

sizes likely helps because it removes the situation when certain wires with a comparatively large fanout get too congested as others do not provide enough possibilities to complete the needed paths.

8 MULTIPLEXER INPUT SHARING

Certain commercial FPGAs employ switch-patterns where pairs of multiplexers share a large fraction of inputs. For example, Young showed that forming a pair of two 6:1 multiplexers that share 5/6 inputs allows for significant area gains due to diffusion sharing [1, 14]. Similarly, Chromczak et al. state that "adjacent muxes aggressively share input pins" to reduce vias in Intel Agilex FPGAs [8]. In this section, we measure the impact of this design approach.

8.1 Encoding

To begin encoding the constraints that will force the multiplexers to form pairs which share the required number of inputs, we first track the number of inputs shared by each pair of wires, v_1 and v_2

$$S_{v_1,v_2} = \sum_{u,d_L:(\forall v \in \{v_1,v_2\})((u,v,d_L)\in E)} x_{u,v_1,d_L} \wedge x_{u,v_2,d_L}, \quad \forall v_1, v_2 \in V, \tag{12}$$

where S_{v_1,v_2} is an additional variable that counts the number of inputs shared between the wires v_1 and v_2. To linearize the conjunction, we need to introduce another binary variable x_{u,v_1,v_2,d_L} for each pair of wires v_1 and v_2 and each input switch that they could share (each addend of the above sum). The conjunction is then linearlized in a standard way by the following constraints [15]:

$$x_{u,v_1,v_2,d_L} \geq x_{u,v_1,d_L} + x_{u,v_2,d_L} - 1, \tag{13}$$

$$x_{u,v_1,v_2,d_L} \leq x_{u,v,d_L}, \quad \forall v \in \{v_1, v_2\} \tag{14}$$

Next, for each pair of wires v_1 and v_2, we introduce a binary variable F_{v_1,v_2} which, when 1, will force v_1 and v_2 to share at least ξ inputs, where ξ is the specified constant:

$$\xi F_{v_1,v_2} \leq S_{v_1,v_2}, \quad \forall v_1, v_2 \in V. \tag{15}$$

Finally, we need to partition the wires into pairs, by specifying that each wire is forced into exactly one pair:

$$\sum_{v_2 \in V} F_{v_1,v_2} = 1, \quad \forall v_1 \in V. \tag{16}$$

8.2 Results

In these experiments, we force all multiplexers to take six inputs from other wires and sweep the minimum sharing bound ξ between 0 and 5. Since there are 16 wires in the switch-pattern, the total number of switches is exactly 96 and the multiplexer and fanout sizes correspond to the uniform case ($N_\varphi = 1$) of Section 7. The multiplexers are not exactly 6:1 like those of Young [14], since each takes two additional inputs from the LUTs.

Critical Path Delay: Influence of input sharing between pairs of multiplexers on critical path delay of routed circuits is shown in Figure 8. No significant critical path delay increase is apparent. In fact, there even appear to be some gains from applying these constraints, likely because wiring load is reduced. This can be observed in Table 2 which contains average total wire delays for architectures with different amounts of input sharing. It is important to note that the models which we use [17] are not capable of accounting for area reduction obtained through diffusion sharing, nor for the reduced via use, which were the primary reasons for input sharing stated by Young [14] and Chromczak et al. [8]. Since input sharing appears to make no negative impact on the routed critical path delays, even with the conservative models that only take into account reduction in length of wires feeding multiplexer inputs, we hypothesize that FPGA vendors are reaping significant gains from this technique when its impact on layout is fully taken into account.

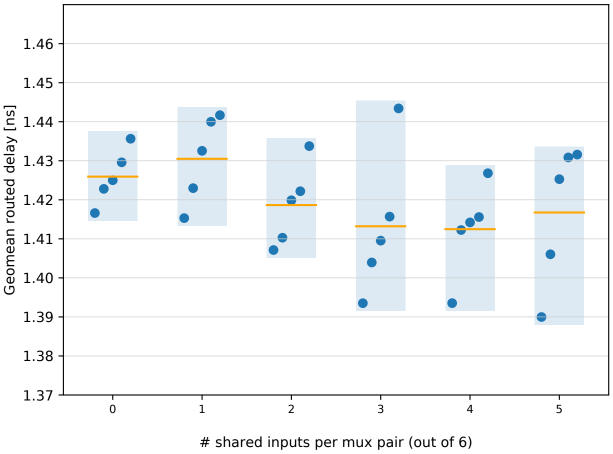


Figure 8: Influence of sharing inputs between pairs of multiplexers on critical path delay. Input sharing shows no significant negative impact on critical path delay; it even slightly improves it in some cases.

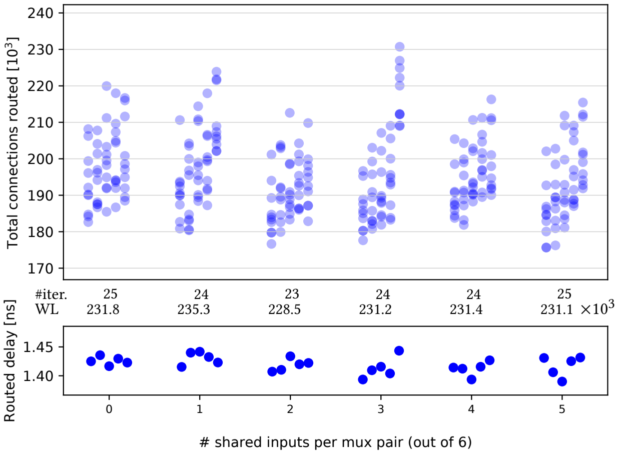


Figure 9: Influence of sharing inputs between pairs of multiplexers on routability. Input sharing shows no negative impact on routability.

Routability: One could expect that sharing majority of inputs between pairs of multiplexers may cause a drop in routability. However, the results of Figure 9 show that this is not the case. Combining this observation with the expected performance gains, area reduction, and easier layout that were previously discussed, we conclude that input sharing is likely a very effective optimization technique.

9 MINIMIZING WIRELENGTH

So far, we have only used total switch type usage as the ILP problem objective. However, it may happen that some of the switch types are added to the switch-pattern solely to make it regular and that they otherwise have very low and even zero usage. In fact, in the experiments presented in this work, due to small size of the circuits on which the search is conducted, it was not uncommon to observe > 20 of the 96 switch types entering the pattern exclusively for regularization reasons. In such cases, it seems reasonable to have a complementary objective to guide regularization. A good candidate for this is minimizing the total length of the wires providing inputs to the multiplexers in the switch-block. In this section, we describe how that objective can be modeled and what impact it has on performance and routability of the obtained solutions.

9.1 Encoding: Modeling Wirelength

Without loss of generality, we assume that all multiplexers are of the same size and that their physical width and height are α and β, respectively. For the solving approach discussed shortly, which formulates and solves multiple ILP problems with fixed multiplexer placement, relaxing this constraint is trivial. Same-size multiplexers form a uniform grid of n columns and m rows. We represent the placement of the different multiplexers in a similar way as Mihal and Teig [18]. For each wire $v \in V$, and each location on the uniform grid (x, y), a binary variable $x_{v,(x,y)}$ indicates that the multiplexer driving v is placed at location (x, y). The following constraints make sure that each multiplexer is assigned a unique location:

$$\sum_{x,y} x_{v,(x,y)} = 1, \qquad \forall v \in V. \tag{17}$$

Overlaps between multiplexers are removed as follows:

$$\sum_{v \in V} x_{v,(x,y)} \leq 1, \qquad \forall (x, y). \tag{18}$$

We describe the total wirelength as the sum of Manhattan lengths of individual switch types:

$$\sum_{\substack{(u,v,d_L) \in E \\ x',x" \in [0,n) \\ y',y" \in [0,m)}} \lambda(x', y', x", y", d_L)(x_{u,v,d_L} \wedge x_{u,(x',y')} \wedge x_{v,(x",y")}), \tag{19}$$

where $\lambda(x', y', x", y", d_L)$ is a precomputed constant Manhattan distance between locations $(x', x")$ and $(y', y")$. The conjunction is linearized as in Section 8.1. To reduce solution time, instead of allowing the ILP solver to assign the locations of the multiplexers, we fix them a priori, then solve the simpler ILP problems and reiterate the process several times (five in the experiments presented here), optimizing the placement between iterations using simulated annealing. During placement optimization, switch type selection is considered fixed, as determined by the ILP solution. We use the annealing schedule of Betz et al. [19]. Tighter formulations of multiplexer placement—essentially a *Quadratic Assignment Problem* with Manhattan distance—is possible [20, 21] and could potentially allow the ILP solver to simultaneously select switch types and place multiplexers. However, this exceeds the scope of the present work. We note that multiplexer position optimization is performed in every iteration of the algorithm of Figure 3, prior to SPICE simulations [2]. Additional optimization performed here is only intended to allow the ILP solver to form better decisions with respect to wirelength minimization and does not constitute an advantage in its own right.

9.2 Encoding: Combined Objective

To combine wirelength minimization with usage maximization, we adapt the *auto-normalizing* function of Marquardt et al. [22]:

$$\min \left(\Lambda \frac{\text{total WL}}{\text{previous total WL}} - (1 - \Lambda) \frac{\text{total U}}{\text{previuos total U}} \right). \tag{20}$$

Objectives *total WL* and *total U* are determined by equations (19) and (3), respectively, while *previous total WL* and *previous total U* designate the total wirelength and usage of the ILP solution from the previous iteration. Before solving the first ILP problem, a greedy solution is formed by selecting the M (upper bound on switch-pattern size, see Section 4.2) most used switch types, from which the initial *previous total WL* and *previous total U* are determined. Constant Λ sets the *wirelength trade-off.*

9.3 Results

In this section we present the results of experiments where all wires were constrained to have the same fanout (six, disregarding the connection block) and the same fanin (six, disregarding the LUT drivers). No other constraints were imposed and the wirelength trade-off constant, Λ, was swept from 0 (wirelength completely disregarded) to 1 (usage completely disregarded).

Critical Path Delay: Figure 10 shows the impact of minimizing wirelength on critical path delay of the implemented circuits. We see by comparing the edge cases ($\Lambda = 0$ and $\Lambda = 1$) that when the multiplexers are reasonably large and fanouts and fanins are uniform, which helps eliminate any pathological solutions, optimizing wirelength is about as important as respecting the affinity of the router towards certain switch types. The best results arise when both objectives are combined, however. The exact value of the trade-off constant Λ does not seem to matter too much, but the plot suggests that setting it to 0.5 yields the best results.

Routability: Figure 11 indicates that taking wirelength into account also slightly improves routability. One hypothesis why that is so is that better distribution of capacitive loads alters the base costs of routing resources assigned by VPR [11], leading to reduced congestion. Since the differences are not as obvious as with delays, we did not attempt to put this hypothesis to test yet.

10 ENFORCING TURNS AND SYMMETRIES

So far, all the constraints that we have presented were motivated by layout considerations. In this section we investigate several other constraint types that should not directly impact the ease with which the switch-blocks can be laid out on silicon, but they do reflect features commonly observed in both industrial and academic architectures and may impact performance of the CAD tools.

10.1 Encoding: Turns

The first set of constraints is inspired by the assumption that has been very popular among the academic architectures since it was first formulated by Rose and Brown [23]: namely, that each wire should fan out to one wire on each of the tree remaining sides of the switch-block; the two that are perpendicular to where the wire

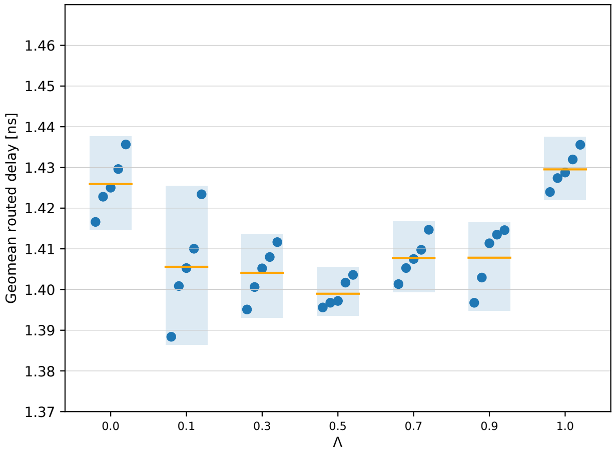


Figure 10: Influence of wirelength optimization on critical path delay of implemented circuits. $\Lambda = 0$ only maximizes usage, while $\Lambda = 1$ only minimizes wirelength. Joint optimization of both objectives leads to the best results.

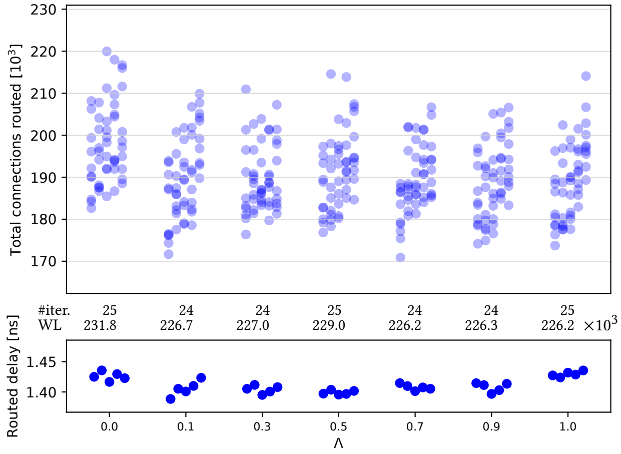


Figure 11: Influence of wirelength optimization on routability. Joint optimization of wirelength and usage also leads to a slight increase in routability.

came from and the one that is directly opposite of it. We generalize this constraint to allow for larger multiplexers and larger fanouts, by turning the *exactly one* requirement to *at least one* requirement:

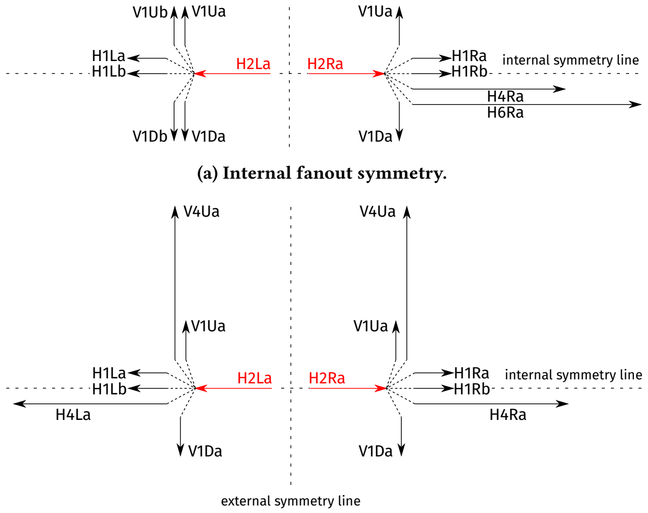
$$\sum_{v,d_L:(u,v,d_L)\in E\wedge DIR(v)=f_L(DIR(u))} x_{u,v,d_L} \geq 1, \qquad \forall u \in V, \tag{21}$$

$$\sum_{v,d_L:(u,v,d_L)\in E\wedge DIR(v)=f_R(DIR(u))} x_{u,v,d_L} \geq 1, \qquad \forall u \in V, \tag{22}$$

$$\sum_{v,d_L:(u,v,d_L)\in E\wedge DIR(v)=DIR(u)} x_{u,v,d_L} \geq 1, \qquad \forall u \in V, \tag{23}$$

$$f_L = \begin{pmatrix} R & L & U & D \\ U & D & L & R \end{pmatrix}, \qquad f_R = \begin{pmatrix} R & L & U & D \\ D & U & R & L \end{pmatrix}. \tag{24}$$

Equations (21), (22), and (23) specify that each wire must turn left, turn right, and continue in the same direction, respectively. Function $DIR(v)$ returns the direction of wire v, while functions f_L and f_R describe the mappings between directions that constitute left and right turns, respectively.



(a) Internal fanout symmetry.

(b) External fanout symmetry.

Figure 12: Fanout symmetry illustration. Fanouts of the *H2Ra* and *H2La* wires (red) are shown for two different patterns. (a) Each of the wires has a symmetric fanout: there is a mirror symmetry between the vertical wires in the fanout, but the fanouts of the two wires are not mutually symmetric. (b) The two wires have mutually symmetric fanouts: the set of vertical wires they fanout to is identical, while there is mirror symmetry between the horizontal wires in the fanout. However, neither of them has an internally symmetric fanout.

10.2 Encoding: Fanout Symmetries

The second set of constraints forces wire fanouts to be symmetric. There are two kinds of symmetry that we explore. The first one which we call *internal symmetry*, illustrated in Figure 12a, specifies that whenever a wire u drives a wire v perpendicular to it, it must also drive a wire v' going in the opposite direction from v. We call the second kind *external symmetry* and it is illustrated in Figure 12b. It specifies that two wires which differ only in direction, such that they are opposing, must have identical sets of perpendicular wires to which they fan out and that the sets of parallel wires they fan out to must differ only in direction (i.e., the directions of the respective wires must be opposing). Both internal and external symmetry are trivially encoded by equating the switch-type-presence variables of symmetric pairs of switch types (e.g., $x_{H2La,V1Ua} = x_{H2La,V1Da}$ in Figure 12a or $x_{H2La,V1Ua} = x_{H2Ra,V1Ua}$ in Figure 12b). In the interest of space, we do not present them here in their general form.

10.3 Results

All experiments in this section were conducted using an objective that combines usage maximization and wirelength minimization with a trade-off of 0.5, since this was previously shown to yield the best results. Fanins and fanouts of all wires were fixed at 6 (disregarding LUT-related ones) and no additional constraints were applied other than the ones indicated in the respective plot.

Critical Path Delay: As shown by Figure 13, enforcing any of the constraints from this section brings no benefit to the critical path delay and even slightly deteriorates it.

Routability: Figure 14 shows that most constraints do not bring a tangible improvement of routability either. Hence, we conclude that there is no benefit in explicitly enforcing them, unless maybe when CAD tools specifically assume that they are satisfied.

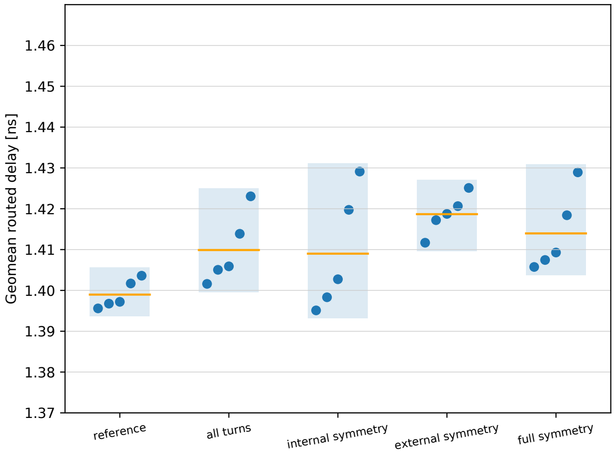


Figure 13: Influence of enforcing different topological features on delay. Enforcing any of these features leads to a deterioration in performance.

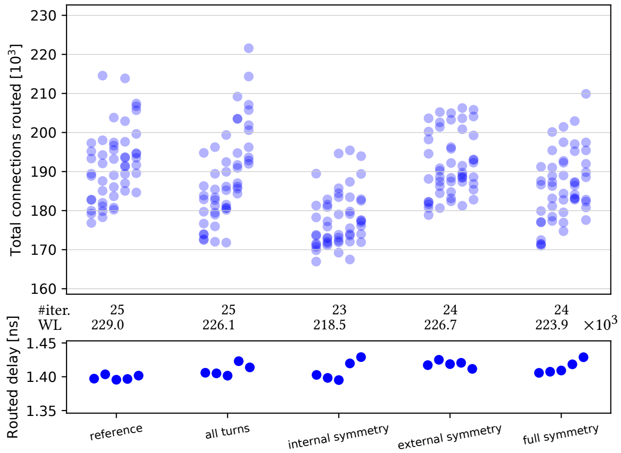


Figure 14: Influence of enforcing different topological features on routability. Enforcing most of the considered features brings little change.

11 ENFORCING HOP-DISTANCE OPTIMALITY

The final set of constraints that we consider is also related solely to the performance of the CAD tools and was inspired by our previous observation that performance of the switch-pattern might be related to the lower bound on the number of hops needed to connect any two locations on the FPGA grid [2]. It could be tempting to search for only those solutions for which this lower bound is the same as if all switch types were present in the pattern.

11.1 Encoding: Proof Grid

We start by computing the length of the shortest path from the center of the FPGA grid to all locations that are at most 12 tiles away from it horizontally, and at most 8 tiles away from it vertically, when all switch types are present in the switch-pattern. The dimensions are dictated by twice the length of the longest horizontal (M_H) and vertical (M_V) wires, respectively (Figure 1b). Without attempting a formal proof, we assume that proving that the same shortest path lengths are attainable by a pattern SOL on this grid suffices to prove that they are attainable by it on an infinite grid, which would render it *hop-distance optimal*. We experimentally confirmed hop-distance optimality of all solutions on a 100 × 100 grid.

11.2 Encoding: Shortest Paths

For the switch-pattern to be optimal, for each offset (x_δ, y_δ), the length of the shortest path connecting the tiles at this offset must equal the value precomputed above, $L(x_\delta, y_\delta)$. Hence, we need to encode an existence of a path of length $L(x_\delta, y_\delta)$ such that the sum of offsets of all horizontal (vertical) wires on it equals x_δ (y_δ). We do this similarly to how Hamiltonian Path can be encoded in SAT [24]: for each position on the path, $p \in [1, L(x_\delta, y_\delta)]$, and each wire v, a binary variable $x_{(x_\delta, y_\delta), p, v}$ is 1 iff v is the p^{th} node on the path. Each position on the path must be occupied by exactly one wire:

$$\sum_{v \in V} x_{(x_\delta, y_\delta), p, v} = 1, \qquad \forall p \in [1, L(x_\delta, y_\delta)]. \tag{25}$$

For two wires to be on consecutive positions in the path, there needs to be a switch between them:

$$\begin{gathered} x_{(x_\delta, y_\delta), p, u} \wedge x_{(x_\delta, y_\delta), p+1, v} \leq \sum_{d_L} x_{u, v, d_L}, \\ \forall p \in [1, L(x_\delta, y_\delta)), \forall u, v \in V. \end{gathered} \tag{26}$$

It only remains to sum up the horizontal wire offsets along the path and force them to x_δ (vertical offsets are analogous):

$$\sum_{p \in [1, L(x_\delta, y_\delta)], v \in V} \chi(v) x_{(x_\delta, y_\delta), p, v} = x_\delta, \tag{27}$$

where $\chi(v)$ is the horizontal offset of the wire v. To speed up the solution process, we add some additional constraints. Most importantly, if the number of shortest paths for a particular offset in the presence of all switch types is less than 100, we enumerate all of them and assign each another binary variable. Sum of all these path variables for the given offset is set to equal 1, which forces the solver to select exactly one of the paths. Then, the corresponding $x_{(x_\delta, y_\delta), p, v}$ variables are set by appropriate implication constraints.

11.3 Results

The experimental setup is the same as in Section 10, but because hop-distance optimality constraints make the problem significantly more complex and increase the solution time from the order of seconds to the order of minutes, we decided to set the wirelength trade-off to 0, thus avoiding the need to solve multiple ILP problems per iteration, due to multiplexer placement optimization.

Critical Path Delay: Figure 15 indicates that enforcing hop-distance optimality slightly improves the performance of the obtained architectures, although the difference is rather insignificant.

Routability: Similar conclusions about the impact that hop-distance optimality has on routability can be drawn from the results shown in Figure 16. Combining the two results, we conclude that imposing these constraints is not detrimental; the guaranteed optimality could perhaps be useful for simplifying the CAD tools.

12 ILP COMPLEXITY

Most of the problem instances that we encountered are solved within second. A notable exception are the problems which enforce hop-distance optimality that sometimes take minutes and even tens of minutes to solve. Comparatively difficult instances also

Table 3: ILP complexity.

	variables		constraints		
reg. type	general	concrete	general	concrete	avg. sol. t [s]
# fanins (fanouts)	$(M_\varphi + 1)(\lvert V\rvert + 1)$	336	$(M_\varphi + 1)(\lvert V\rvert + 1) + 2\,\lvert V\rvert$	368	7.18
input sharing	$\frac{3}{2}\lvert V\rvert^3 + \lvert V\rvert^2$	3 174	$\frac{9}{2}\lvert V\rvert^3 + \lvert V\rvert^2 + \lvert V\rvert$	9 418	40.66
all turns	0	0	$3\,\lvert V\rvert$	48	1.32
internal (external) symmetry	0	0	$\frac{3}{2}\lvert V\rvert^2$	282	2.68
full symmetry	0	0	$3\lvert V\rvert^2$	564	2.67
hop-distance optimality	$(4M_H + 1)(4M_V + 1)2(M_H + M_V)\,\lvert V\rvert$	92 469	$(4M_H + 1)(4M_V + 1)(2(M_H + M_V)\lvert V\rvert^2 + 1)$	358 846	161.40

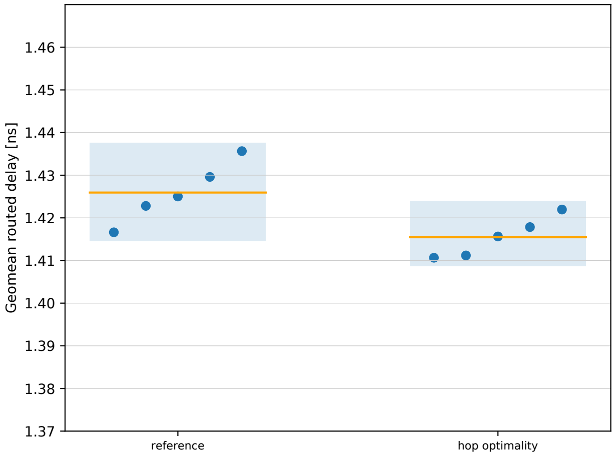


Figure 15: Influence of hop-distance optimality on critical path delay.

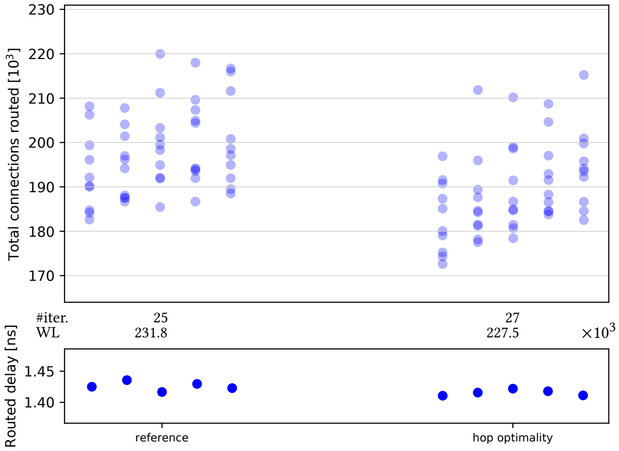


Figure 16: Influence of hop-distance optimality on routability.

occur when sharing of a large number of inputs is enforced, where solution times sometime also reach minutes.

In Table 3, we give average solution times as well as the number of variables and constraints needed to encode various types of regularity introduced in previous sections. All of the formulas are upper bounds. Given that modern architectures, faced with numerous technological limitations, have highly repetitive interconnect (e.g., *planes* and *lanes* in Agilex [8]), the number of multiplexers that need to participate in the ILP problems is unlikely to ever become very large. Thus, we expect the proposed method to be scalable even for regularity types for which the encoding size increases cubically with multiplexer count. Moreover, ILP problem size does not depend on the size of the circuits used in the exploration, unlike avalanche routing, which is at present the runtime bottleneck of the exploration algorithm [2].

13 REGULARITY MATTERS

We have shown that various types of regularity observed in the switch-blocks of commercial FPGAs do not pose a significant limitation on their performance. In particular, the best-performing architecture where all wires are forced to have the same fanout and fanin size ($\Lambda = 0.5$ in Figure 10) has only about 1.5% worse routed critical path delay than the best architecture constructed when the router is free to choose switch types according to its needs ("∞" in Figure 4). On the other hand, regularity in fanout and fanin sizes greatly increases the routability of the architectures on circuits that significantly differ from the ones used in switch-pattern construction (Figure 7). This is certainly important for FPGA vendors, given the long lifetime of the products and difficulties in predicting how much the target designs will change over that period.

Regularity of the switch-pattern may allow layout optimizations that are not captured by the existing academic area and delay models. Hence, the measured loss of performance due to regularization could be even smaller in a commercial setting. For example, we demonstrated that sharing inputs between pairs of multiplexers does not have any negative impact on routability (Figure 9) and that it even provides a slight advantage in terms of performance when measured using conservative academic models (Figure 8). On the other hand, several authors have mentioned other reasons why this technique can be beneficial [8, 14], which, when combined with the above result, may more than suffice for the FPGA architect to seek only switch-patterns where all multiplexers share some inputs.

Perhaps most importantly, we have demonstrated that automated exploration methods can be used to construct competitive switch-block architectures that respect arbitrary constraints, ranging from fairly simple ones motivated by layout considerations, such as those presented in Section 6, to fairly complex ones motivated by design ideas that may enable more efficient FPGA CAD, presented in Section 11. Given that the algorithm which we proposed is generic, our hope is that it will be useful for both industry and academia in addressing the challenges that designing future FPGAs may bring.

The source code used in this work is available at the following link: https://github.com/EPFL-LAP/fpga23-regularity.

14 ACKNOWLEDGEMENTS

We would like to thank Jason Anderson for fruitful discussions as well as the anonymous reviewers for their constructive feedback which greatly improved the quality of this work.

REFERENCES

[1] M. B. Petersen, S. Nikolić, and M. Stojilović, “NetCracker: A peek into the routing architecture of Xilinx 7-Series FPGAs,” in *Proceedings of the 2021 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, Virtual Event, Feb. 2021, pp. 11–22.

[2] S. Nikolić and P. Ienne, “Turning PathFinder upside-down: Exploring FPGA switch-blocks by negotiating switch presence,” in *Proceedings of the 31st International Conference on Field-Programmable Logic and Applications*, Dresden, Aug. 2021, pp. 225–33.

[3] M. Lin, J. Wawrzynek, and A. E. Gamal, “Exploring FPGA routing architecture stochastically,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 10, pp. 1509–22, 2010.

[4] J. Qian, Y. Shen, K. Shi, H. Zhou, and L. Wang, “General routing architecture modelling and exploration for modern FPGAs,” in *Proceedings of the 2021 International Conference on Field-Programmable Technology*, Auckland, Dec. 2021, pp. 1–9.

[5] G. Lemieux, P. Leventis, and D. Lewis, “Generating highly-routable sparse crossbars for PLDs,” in *Proceedings of the 2000 ACM/SIGDA Eighth International Symposium on Field Programmable Gate Arrays*, Monterey, Calif., Feb. 2000, pp. 155–64.

[6] D. Lewis, V. Betz, D. Jefferson, A. Lee, C. Lane, P. Leventis, S. Marquardt, C. McClintock, B. Pedersen, G. Powell, S. Reddy, C. Wysocki, R. Cliff, and J. Rose, “The Stratix routing and logic architecture,” in *Proceedings of the 2003 ACM/SIGDA Eleventh International Symposium on Field Programmable Gate Arrays*, Monterey, Calif., Feb. 2003, pp. 12–20.

[7] D. Lewis, D. Cashman, M. Chan, J. Chromczak, G. Lai, A. Lee, T. Vanderhoek, and H. Yu, “Architectural enhancements in Stratix V™,” in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, Monterey, Calif., Feb. 2013, pp. 147–56.

[8] J. Chromczak, M. Wheeler, C. Chiasson, D. How, M. Langhammer, T. Vanderhoek, G. Zgheib, and I. Ganusov, “Architectural enhancements in Intel® Agilex™ FPGAs,” in *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, Seaside, Calif., Feb. 2020, p. 140–49.

[9] R. Rubin and A. DeHon, “Timing-driven PathFinder pathology and remediation: Quantifying and reducing delay noise in VPR-PathFinder,” in *Proceedings of the ACM/SIGDA 19th International Symposium on Field Programmable Gate Arrays*, Monterey, Calif., Feb. 2011, pp. 173–76.

[10] Y. Zha and J. Li, “Revisiting PathFinder routing algorithm,” in *Proceedings of the 2022 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, Virtual Event, 2022, pp. 24–34.

[11] K. E. Murray, O. Petelin, S. Zhong, J. M. Wang, M. Eldafrawy, J.-P. Legault, E. Sha, A. G. Graham, J. Wu, M. J. P. Walker, H. Zeng, P. Patros, J. Luu, K. B. Kent, and V. Betz, “VTR 8: High-performance CAD and customizible FPGA architecture modelling,” *ACM Transactions on Reconfigurable Technology and Systems*, vol. 13, no. 2, pp. 9:1–60, May 2020.

[12] J. Anderson and F. Najm, “Active leakage power optimization for FPGAs,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 3, pp. 423–37, 2006.

[13] C. Wolf and M. Lasser, “Project icestorm,” http://bygone.clairexen.net/icestorm/.

[14] S. P. Young, “Six-input multiplexer with two gate levels and three memory cells,” US Patent 5 744 995, 1998.

[15] H. P. Williams, *Model Building in Mathematical Programming*, 5th ed. Wiley, 2013.

[16] D. Stroobandt, J. Depreitere, and J. V. Campenhout, “Generating new benchmark designs using a multi-terminal net model,” *Integration*, vol. 27, no. 2, pp. 113–29, 1999.

[17] S. Nikolić, F. Catthoor, Z. Tőkei, and P. Ienne, “Global is the new local: FPGA architecture at 5nm and beyond,” in *Proceedings of the 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, Virtual Event, Feb. 2021, pp. 34–44.

[18] A. Mihal and S. Teig, “A constraint satisfaction approach for programmable logic detailed placement,” in *Proceedings of the 16th International Conference on Theory and Applications of Satisfiability Testing*, Helsinki, Jul. 2013, pp. 208–23.

[19] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer Academic Publishers, 1999.

[20] E. L. Lawler, “The quadratic assignment problem,” *Management Science*, vol. 9, no. 4, pp. 586–99, 1963.

[21] S. Gueye and P. Michelon, “A linear formulation with $O(n^2)$ variables for quadratic assignment problems with manhattan distance matrices,” *EURO Journal on Computational Optimization*, vol. 3, no. 2, pp. 79–110, 2015.

[22] A. Marquardt, V. Betz, and J. Rose, “Timing-driven placement for FPGAs,” in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, Monterey, Calif., Feb. 2000, pp. 203–13.

[23] J. Rose and S. Brown, “Flexibility of interconnection structures for field-programmable gate arrays,” *IEEE Journal of Solid-State Circuits*, vol. 26, no. 3, pp. 277–282, 1991.

[24] C. Papadimitriou, *Computational Complexity*. Addison-Wesley, 1994.