NetCracker: A Peek into the Routing Architecture of Xilinx 7-Series FPGAs

Morten B. Petersen, Stefan Nikolić and Mirjana Stojilović
EPFL, School of Computer and Communication Sciences, CH-1015 Lausanne, Switzerland

ABSTRACT

Novel applications have triggered significant changes at the system level of FPGA architecture design, such as the introduction of embedded VLIW processor arrays and hardened NoCs. However, the routing architecture of the soft logic fabric has largely remained unchanged in recent years. Since hunger for acceleration of ever more varied tasks with various power budgets—as well as complications related to technology scaling—is likely to remain significant, it is foreseeable that the routing architecture too will have to evolve. In this work, we do not try to suggest how routing architectures of tomorrow should look like. Instead, we analyze an existing architecture from a popular commercial FPGA family, discussing the possible origins of various design decisions and pointing out aspects that may merit future research. Moreover, we present an open-source tool that greatly eases such analyses, relying only on data readily available from the vendor CAD tools. Our hope is that this work will help the academic research community in catching up with the current developments in industry and accelerate its contributions to FPGA architectures of the future.

ACM Reference Format:
Morten B. Petersen, Stefan Nikolić and Mirjana Stojilović. 2021. 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 (FPGA '21), February 28-March 2, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3431920.3439285

1 INTRODUCTION

Three decades ago, there were a plethora of FPGA vendors with vastly different architectures [1]. It was only the comprehensive studies performed by the academic research community that gave answers to such questions as whether it is better to base the logic cell on a *Look-Up Table* (LUT) or an And-Or gate [2], which are now taken for granted. Today, but a few vendors remain in business, programmable fabric becomes a smaller and smaller part of the FPGA [3, 4], and we largely believe that we have a solid understanding of how it should be designed [5]. But, do we?

Perhaps it is not too surprising that whether a DSP block has a floating point unit or not can make a big difference in how well the FPGA performs certain tasks [6], but that architectures with a sparse cluster input crossbar [7, 8] and without any such crossbar at all [9] can perform similarly well should probably not be dismissed as a fact of life. When such major differences appear between the programmable fabric architectures of the two major FPGA vendors—Intel and Xilinx—we cannot but wonder *why that is so?*

There exists a substantial body of academic research assessing various trade-offs that impact the performance of FPGA architectures resembling the Stratix family of Intel [10]. The series of papers presenting the Stratix architectures [7, 8, 11–13] in turn reveals that they have been heavily influenced by prior work in academia, in particular the VPR project from the University of Toronto [7, 14].

Unfortunately, to the best of our knowledge, there has been comparably little recent research that explored the trade-offs in Xilinx-like architectures, let alone between them and the Intel-like ones [15, 16]. This means that a significant portion of knowledge related to FPGA architecture design that already exists and that could be of great value in solving future challenges remains unexplored by the wider community.

Systematic analysis of various trade-offs to find the best set of choices that may enter the next FPGA architecture goes beyond the scope of this paper. Instead, we start from the premise that before an attempt at such an analysis is made, one should understand the existing architectures that are already proven to be successful.

To facilitate the analysis of an FPGA routing architecture, we developed *NetCracker*: a flexible framework consisting of several built-in passes for various kinds of combinatorial and statistical analysis of the routing network, which can be easily complemented by additional user-written passes. NetCracker is fully vendor-agnostic; it requires only the information about the connections between FPGA resources, often readily available from the design tools.

NetCracker is one of our two main contributions, presented in Section 4. The other is applying it to the 7-Series architecture family, the results of which we report in Sections 5, 6, and 7. Apart from demonstrating the capabilities of NetCracker, the goal of this analysis is to identify the similarities with and differences from the typical assumptions in academia and the public information about the Intel architectures, and to suggest possible reasons that may have driven such design choices.

We note here that all the presented conclusions have been obtained by analyzing information readily available from Xilinx design tools and that we have not had any support in performing the analysis from the company itself. Hence, the discussion about the possible reasons behind the design choices is not to be understood as factual, but as our best effort in interpreting the observations using results from prior work. The sole purpose of the analysis is to help shrink the gap between academic research and industrial reality, and point to some potentially interesting research problems. The choice of analyzing the 7-Series architecture in particular is due to it having recently positioned itself as the first high-end family targeted by a fully open-source CAD flow [17]. This likely means that the community will show the most interest in understanding its architecture at the present moment.

FPGA '21, February 28-March 2, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8218-2/21/02.
https://doi.org/10.1145/3431920.3439285

2 RELATED WORK

Substantial amount of detail about the low-level architecture of the newer Xilinx chip families has been published in prior work presenting the state-of-the-art CAD algorithm research results, coming both from the company itself [18–20], as well as the academic community participating in Xilinx-organized contests [21, 22]. These contests helped in raising the interest for solving real-world problems, related to commercial architectures and considering restrictions and optimization goals previously largely neglected in academia. To the best of our knowledge, however, they did little to inspire new architectural research. Rather than discussing the various considerations that entered the design of the particular architecture, they only present the details necessary for the algorithmic problems to be well specified, whereafter these details are considered merely as given facts. While papers focusing on architecture itself do exist [4, 23], the attention they give to the reasons behind the routing structure having taken the shape that it has falls short of that present in the Stratix and Agilex paper series [7, 8, 11–13, 24, 25], which may partially explain the comparative dominance of Intel-like architectures in academic FPGA architectural research.

Even more detail is required to program a physical chip, than to provide legal solutions to synthesis and physical implementation problems. Some of it is available in official documentation [26], more can be queried from tools [27], while the rest can be retrieved through meticulous analysis of correlation between carefully perturbed microbenchmarks and the bitstream files generated after their implementation [28]. The ability to program a recent commercial device with custom tools is invaluable for driving algorithmic research forward, so it is not surprising that large academic effort has been put into it [9, 29, 30]. With recent proliferation of FPGAs into datacenters, autonomous vehicles, and mobile devices, security and privacy issues slowly gain in importance as well, which makes it natural for the open source software community to also join the effort [28, 31, 32]. Many of these projects are using VPR [33] for placement and routing, and have thus greatly extended the number of detailed architecture models available in this framework, ubiquitously used for academic research in the field of FPGA architecture and CAD [9, 17].

The detailed architecture models developed for programming commercial devices perfectly suit their purpose: programming. Their availability is not sufficient to drive architectural research, though. For instance, in the early days of FPGAs, when the chip size was still modest and CAD tools still in their infancy, it was not uncommon to resort to manual design to achieve high performance and density [34]. This inevitably meant that low-level architectural details were available to the end user [35]. However, it was only when these details were analyzed, categorized, abstracted, and formalized [36–39] that a general understanding of the various trade-offs started to emerge. In this work, we build infrastructure that makes this task easier, hoping that our conclusions will both contribute to the general understanding of how a good routing architecture may be designed and incite future research in the area.

3 PRELIMINARIES

In 7-Series FPGAs, configurable logic blocks (CLBs), transceivers, DSPs, and other building blocks are arranged in columns. A CLB is composed of two slices, which we will later denote as U and T (Section 7). A slice is composed of four logic elements—A (shown in Fig. 1), B, C, and D—each containing a six-input LUT, two storage elements, wide-function multiplexers, and carry logic.

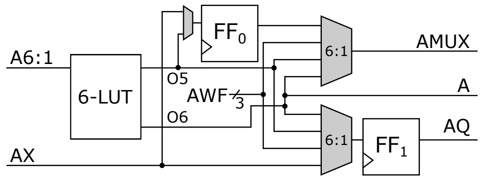


Figure 1: Logic element of a SLICEL [26]. AX input serves to bypass the LUT. Carry and wide-function logic, which also provides the three shared inputs (WF) of the two 6:1 multiplexers, is not shown.

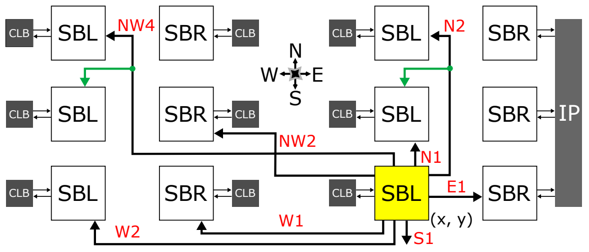


Figure 2: Columnar architecture, with examples of cardinal (horizontal/vertical), noncardinal, and wire stubs (in green). Highlighted in yellow is the source SB; x (resp. y) coordinate grows towards east (resp. north).

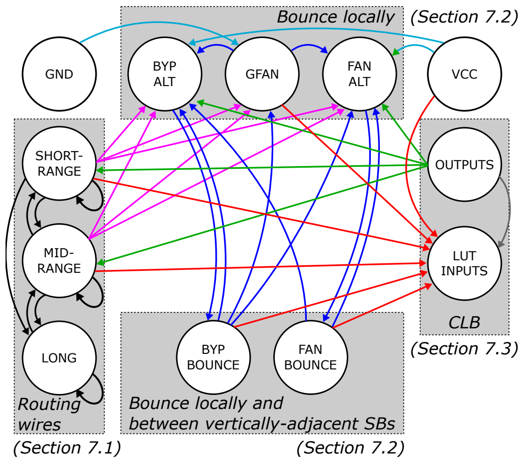


Figure 3: Types of routing multiplexers (PIPJs) inside a switch-box, and the connectivity between them.

Each CLB has a dedicated switch-box (SB). In Vivado, SBs are named either INT_L or INT_R; INT stands for *interconnect*, whereas L (left) and R (right) correspond to the location of the CLB with respect to the SB (Fig. 2). Despite their different names, as we will show in Section 5, these SBs are, in general, identical.

To uncover the routing architecture of 7-Series FPGAs, we choose the Artix-7 XC7A35T as the target device; its relatively small footprint facilitates full-device analysis. Then, we start by selecting an SB and extracting information on the number, length, and direction of all *direct* connections that *emanate* from it. This is illustrated in Fig. 2 and addressed in detail in Section 6. We will see that, besides the horizontal and vertical wires in cardinal directions, there are wires in noncardinal directions—we will refer to them as *diagonal*—and wire stubs (secondary destinations of a wire).

In Vivado, the terminals of global (intercluster) routing wires are named by the wire direction (exclusively either cardinal or intercardinal) and length, i.e., the Manhattan distance between the source and the destination SBs. For example, the terminals of all wires going westwards and spanning four columns will have WW4 in their name. Similarly, the terminals of all wires of length six (LEN-6) going southeast will have SE6 in their name. The drawbacks of this naming scheme are that it hides the information on the precise vertical and horizontal offset between the wire terminals (e.g., SE6 does not uniquely define the x and y offsets) and that it does not capture the location of the secondary destinations. Hence, we introduce a more accurate naming scheme, in which all wire sources and destinations are named using a *direction vector* (x, y), where x (resp. y) is the horizontal (resp. vertical) offset of the destination SB with respect to the source SB. For example, a source (the output of an SB) is named (0, 6) if it drives a wire going north and terminating at an SB six rows away. At the same time, a destination (the input of an SB) is named (0, 6) if it receives a wire coming from south, originating at an SB six rows away.

In Xilinx terminology, routing switches are *programmable interconnect points* (PIPs). A collection of PIPs that drive a signal forms a routing multiplexer, while a collection of routing multiplexers forms a switch-box. A signal can enter or exit an SB through a so-called *PIP junction* (PIPJ).

Analyzing the fan-ins and fan-outs of all PIPJs in a representative SB, we arrive at the diagram shown in Fig. 3. Each node in this graph represents a subset of all PIPJs, grouped by name and functionality; these subsets are disjoint. A directed edge connects two nodes if there exists at least one PIPJ in the first subset that can drive a PIPJ in the other subset. In Fig. 3, we cluster the nodes into three groups: those interfacing routing wires, those interfacing LUT inputs and CLB outputs, and the somewhat special *bounce* PIPJs. In Section 7, we describe and discuss in detail the connections visualized in Fig. 3.

4 NETCRACKER

The analyses performed in this work have been unified into a Python-based tool we name *NetCracker* and make openly available [40]. The architecture of NetCracker is shown in Fig. 4.

At its input, NetCracker expects a file in `json` format, describing a directed graph with switch-box PIPJs as vertices and their incoming and outgoing connections as edges. To add a physical dimension to the graph, the (x, y) coordinates of both the driving and the receiving ends of each edge are required as well. This format is vendor-independent and thus any FPGA routing network adhering to it may be analysed. The data required to create an input file are often readily available in vendor CAD tools.

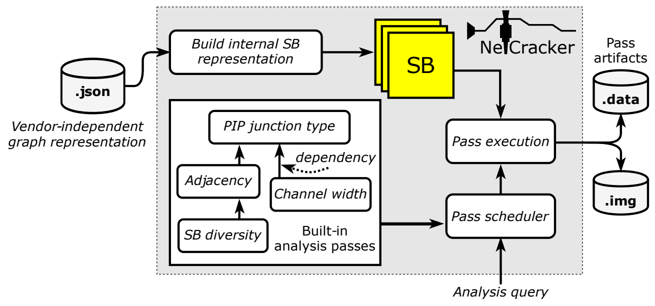


Figure 4: NetCracker architecture.

NetCracker organizes its analyses in a number of *passes* that execute on one or more SBs. Each pass may produce a set of artifacts (plots, statistics, etc.) to facilitate further qualitative analysis by the user, or to enable the execution of other passes. Having each pass being able to produce and consume the results of other passes allows them to be organized in a dependency graph. Given an analysis request, specified by the user through a command-line interface, NetCracker schedules the required set of passes. This implementation approach allows each pass to be as atomic as possible and facilitates extending NetCracker with new analyses.

As an example, given a file `sbs.json`, containing the description of an arbitrary number of SBs, and an analysis query of *SB-diversity* (finding the number and types of different SBs in the input file), the following sequence of operations is executed:

```
sbs = load_switchboxes("sbs.json")
foreach sb in sbs:
   pipj_type(sb)
   adjacency(sb)    # depends on pipj_type
sb_diversity(sbs)   # depends on adjacency
```

Note that pass `sb_diversity` (Section 4.2) runs only once. Whereas `pipj_type` (Section 4.1) and `adjacency` (Section 4.2) are both passes which execute on a single SB, `sb_diversity` executes on all SBs, using the `adjacency` result of each of them.

In what follows, we elaborate on some of the NetCracker built-in passes: the routing channel width and composition pass, the adjacency pass, and the SB-diversity pass.

4.1 Routing Channel Width and Composition

While parsing the input directed graph, NetCracker classifies the PIPJs as inputs, outputs, bidirectional (inputs and outputs) or entirely internal to the SB. This is done in the pass called *PIP junction type* in Fig. 4. Next, it infers the direction vector of every output PIPJ to find the number and length of all wires emanating from the corresponding switch-box.

For finding the channel widths and composition, NetCracker assumes that all SBs have the exact same external connectivity pattern (e.g., all have the same number of LEN-2 wires going south). This assumption, our analyses show, generally holds in 7-Series FPGAs; the exceptions are SBs in the vicinity of hardened resources or device edges. Then, in the first approximation, NetCracker computes the routing channel width as the sum of products of the number of cardinal wires of a given length and their length. To account for the contribution of diagonal wires, NetCracker takes the x and

y components of their direction vector and updates the routing channel widths accordingly. The results of the channel width and composition analysis will be presented and discussed in Section 6.

4.2 Adjacency Analysis Pass

The main feature of NetCracker is the ability to expose the connections between PIPJs of a switch-box at various levels of abstraction. The resulting insight into the topology of the switch-box itself can lead to a better understanding of the motivations behind the switch-box and the routing network design. To do that, NetCracker implements the *adjacency* analysis pass, which can be invoked to inspect the connectivity between either individual PIPJs or between various *sets* of PIPJs. The latter is achieved by labelling and clustering the directed graph describing the switch-boxes.

PIPJ clustering may be vendor-specific or vendor-independent. To implement a Xilinx-specific clustering, we apply some additional knowledge of PIPJ characteristics to the adjacency analysis pass. Specifically, we leverage that the *names* of switch-box PIPJs in Vivado naturally carry distinguishing features, such as wire direction or length, allowing us to implement *index* and *direction* clustering. For example, let us consider the following two sets of PIPJs driving LEN-2 and LEN-4 wires towards north-east: `NE2BEG0`, `NE2BEG1`, `NE2BEG2`, `NE2BEG3` and `NE4BEG0`, `NE4BEG1`, `NE4BEG2`, `NE4BEG3`. Applying the index clustering results in sets `[4]-NE2BEG` and `[4]-NE4BEG`, both of size four (the number inside the square brackets), whereas applying the direction clustering results in a single set `[8]-NE` of size eight. An example of a vendor-independent clustering is *direction-vector* clustering, where all PIPJs with connections external to the SB under analysis (e.g., routing wire sources or destinations) are clustered based on their corresponding direction vector, introduced in Section 3. To avoid ambiguity between the terms direction clustering and direction-vector clustering, we shall refer to the former simply as clustering by *name*.

NetCracker saves adjacency analysis results as matrices, with elements corresponding to the number of connections between individual PIPJs, PIPJ clusters, or a mix of them.

Apart from creating a basis for the visual inspection of the SB topology at various levels of abstraction, adjacency analysis results are used by another NetCracker pass called *switch-box diversity*. In this pass, the adjacency analysis is applied on all input SBs, and its results compared to find differences between them.

5 SWITCH-BOX DIVERSITY

In our first experiment, we query the SB diversity analysis in NetCracker. The results are shown in Fig. 5, where each color, except white, corresponds to a unique switch-box (white regions are free of SBs).

Initially, we exclude the PIPJs connected to long wires (defined in Section 6.3). The results, in left subfigure, show that the SBs connected to IOs, BRAMs, DSPs, or other resources different than CLBs, are not identical to the SBs connected to CLBs (in yellow). The reason for this is that PIPJs normally interfacing CLB inputs and outputs are often unused when the SB is not connected to a CLB. This same figure confirms that the left and right SBs in Fig. 2 are identical, as mentioned in Section 3. Additionally, contrary to the VTR model of Intel Stratix IV architecture [33, 42], we observe no

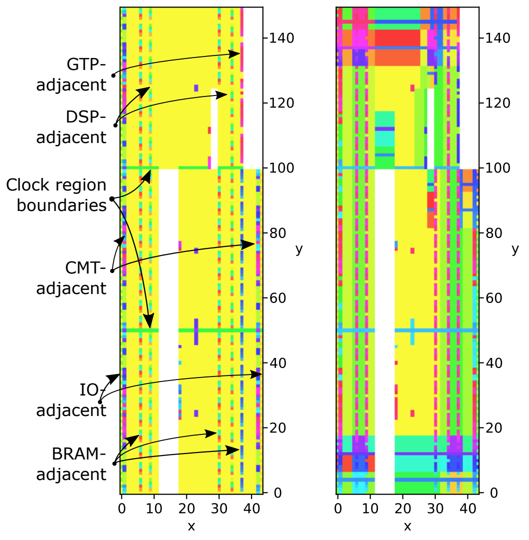


Figure 5: Distribution of switch-box types across the Artix-7 XC7A35T FPGA [41]. Each color represents a unique switch-box. White regions are free of switch-boxes. In the left (resp. right) subfigure, long-wire PIPJs are excluded from (resp. included in) the analysis.

periodic pattern of alternating types of SBs. After this, we include the long-wire PIPJs (right subfigure). The change of colors on all four sides of the device signals that SBs adapt to long wires reaching the edges of the device.

For the analysis of the channel width and composition and the SB internal connectivity (Sections 6 and 7), we choose the right SB at location X15Y128. Being among the most numerous SBs (in yellow), the chosen SB is a representative sample of all SBs interfacing CLBs.

6 ROUTING CHANNELS

Typically, routing channel width and composition (the number and length of wires it contains) have been regarded as important characteristics of an FPGA routing architecture. To find them, we supply NetCracker with a representative switch-box of the target FPGA device (Section 5).

6.1 Short-Range and Mid-Range Connections

Fig. 6 shows where, with respect to the source switch-box, LEN-1 wires terminate and how many there are. Looking at its left subfigure—drawn from the name clustering—we see a perfectly symmetric pattern. However, the direction-vector clustering reveals real target locations, exposing a slightly less regular pattern and the presence of stubs. We also note here that the reduced number of connections ending in the vertically adjacent SBs is the result of two PIPJs (`NL1BEG_N3` and `SR1BEG_S0`) whose name would suggest that they drive a LEN-1 wire, whereas they are local routing multiplexers[1], rerouting signals between local PIPJs.

As we will soon see, out of any single wire type, those of LEN-1 are by far the most numerous. This is in a stark contrast with typical

[1]Uncovered by the adjacency analysis pass in NetCracker (Section 4.2).

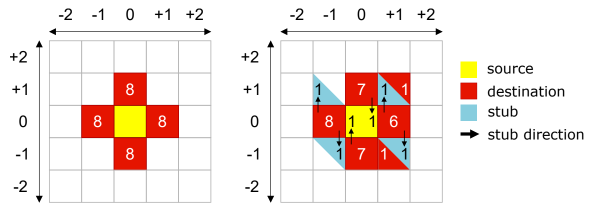


Figure 6: Short-range connections. Left, clustering by name; right, direction-vector clustering. Axes show the offset between the destination and the source SB. The values inside each nonempty cell correspond to the number of connections terminating there.

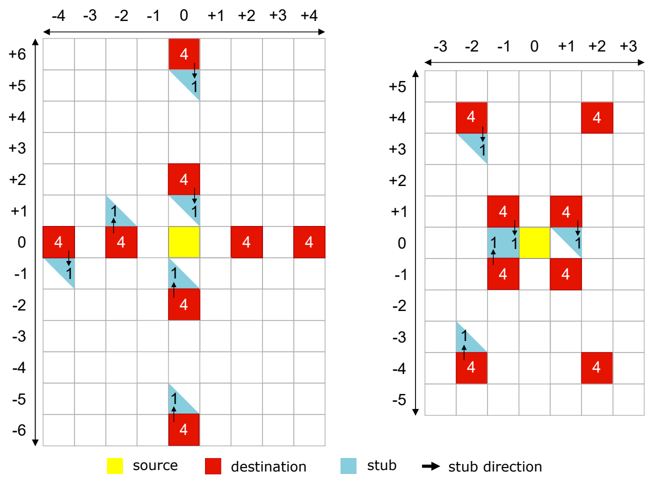


Figure 7: Mid-range connections under direction-vector clustering. Left, cardinal; right, diagonal. Axes show the offset between the destination and the source SB.

academic assumptions that incorporating LEN-1 wires results in a suboptimal design choice [37], as well as with Intel architectures, which incorporate general LEN-1 wires only in the most recent Agilex architecture [8, 25]. The fact that in previous Intel architectures clusters were able to communicate with two adjacent SBs in the same row, as well as that there were direct connections between the outputs of one cluster and the local interconnect of its immediate neighbors [8, 11], could have contributed to this difference.

End locations of the mid-range connections (longer than LEN-1 and shorter than LEN-12) are shown in Fig. 7. We can see perfect symmetry when it comes to the primary targets of both the cardinal and the diagonal connections, even exceeding that of the LEN-1 wires. There is no directional bias in terms of wire counts, which goes in hand with the previous academic conclusions [43]. The span of the vertical wires is slightly larger than that of the horizontal ones (as is the vertical offset of the diagonal wires, compared to their horizontal offset). Assuming that this is an artifact of the CLB tile layout aspect ratio being different from 1:1, as in the Stratix architectures [7], we may suspect that the 7-Series FPGAs have a tile layout whose width exceeds its height.

6.2 Secondary Destinations

So far, our discussion has focused on the primary destinations of each wire. However, there are also the secondary destinations (*stubs*), allowing a wire to branch into an SB other than the one where it finally terminates. Essentially, the stubs are equivalent to the intermediate taps (points that do not coincide with the two ends of the wire, from which a signal can branch to other SBs/connection-boxes, but from which the wire cannot be driven) in academic and Intel architectures. The main difference, however, is that very few wires have a stub (e.g., one in four), while the others have a single termination point. On the other hand, academic architectures typically assume intermediate taps at regular intervals, uniformly specified for all wires of a particular type. The same was true for Intel, until the latest Agilex, which removed taps altogether [25].

Those wires that do have stubs, have them only to an SB immediately adjacent to the terminating one. This eases intermediate buffering of the wire, if applied, as there are no constraints on having an even number of inverters between two taps. If, in the layout, the neighboring SBs are abutted [44], which is likely the case, then the same via can be used to descend through the metal stack for both the terminus and the stub, and what remains is but a very short connection at a lower metal layer, leading to a very small increase in the capacitive load. Speaking of abutting, we should also note that all stubs are designed in such a way that they land in the same column as the primary target. If this were not the case, stubs could not profit from abutting and they would not be as cheap as they likely are with the present situation.

The existence of wire stubs slightly increases the variety of achievable wire lengths in the vertical direction and the number of bends in the horizontal direction. It also allows covering a net with a fanout of two using a single wire, assuming that the placer places the sinks of the same net near each other, which is rather natural. If we observe the distribution of fanout of a typical circuit, we can see that for a vast majority of nets, this will already be sufficient [45], even if they are fully global. This is likely a reason why even some LEN-1 wires have stubs (Fig. 6).

Although it is difficult to say how much exactly an architecture could benefit from such a tapping scheme, we should note the impossibility of arriving at it by exploring only uniform spacing between intermediate taps and varying their count [46], which makes it less surprising that this scheme, to the best of our knowledge, has not yet been explored in an academic setting.

Finally, we should note that another potential reason for existence of a comparatively large number of LEN-1 wires lies precisely in the lack of intermediate taps, which prevents using, e.g., a LEN-6 wire to connect two adjacent SBs. A similar reason recently compelled Intel to introduce such short wires [25].

6.3 Long-Range Connections

The 7-Series FPGAs also contain long wires: LEN-12 in both vertical and horizontal directions, and LEN-18 in vertical directions. This again hints at the thought that the CLB tile is wider than taller.

What is interesting about the long wires is that they appear to be bidirectional, perhaps because they are routed on upper metal layers, where the pitch is wider and, consequently, the space is limited. A similar shift from unidirectional to bidirectional long

Table 1: Horizontal (H) and vertical (V) routing channel width and composition. Asterisk denotes bidirectional wires. Data in the last column considers cardinal wires only.

	Length						Total	Cardinal
	1	2	4	6	12	18	width	only
H	32	48	32	0	12*	0	**124**	74
V	32	16	64	48	12*	18*	**190**	108

wires has already been reported for Intel Arria 10 [24]. This suggests that the typical assumption that unidirectional wires are always superior [47] should probably be revisited in the context of wires whose span substantially exceeds the previously assumed four, with multiplexer-based drivers only at the ends, and high utilization of thicker metal layers.

Long wires are also an exception in terms of intermediate taps: apart from the vertical LEN-12 wires, which have no taps, all the others tap into one switch-box half-way to the end of the wire.

6.4 Channel Widths

For comparison with classical architectures, we report routing channel widths in Table 1. We note, however, that with such a large diversity of the available wire lengths and, with the absence of intermediate taps that prevents using the wires for anything but the connections between their endpoints, the classical notion of channel width likely has little sense: the router simply has much less flexibility in using the six tracks induced by a single LEN-6 wire leaving each SB than it has in using another set of six tracks induced by six individual LEN-1 wires leaving that same SB. Rather than being a design target, the equivalent channel width was likely merely a limiting factor deciding whether a certain set of wire types and counts can actually be implemented. Hence, we believe it is more appropriate to focus on the various wire types and their counts as we did in the previous sections. This is only exacerbated by the presence of diagonal wires. To the best of our knowledge, the concept of diagonal connections, existing at least since Virtex-II (2001) [48], has not yet been a subject of a comprehensive academic study, although similar ideas have been researched [49–51].

7 ADJACENCY ANALYSIS

After seeing the types of connections between switch-boxes, let us now elaborate on the connectivity internal to an SB. We run a number of adjacency analyses in NetCracker and visualize the results as heat maps of the corresponding adjacency matrices (Figs. 8, 10, 11, 12, 13, 14, 16, and 17). For plotting, we use *Morpheus* [52]. In the following subsections, the results are discussed in detail.

7.1 Wire-to-Wire Connections

Fig. 8 shows all possible connections between short- and mid-range wires through the representative SB. A row corresponds to a set of inputs (PIPJs receiving signals); a column corresponds to a set of outputs (PIPJs driving signals). The numbers inside square brackets are the set sizes. Black stands for a complete absence of programmable connections between the input and output sets; red stands for 16, the highest number of connections found.

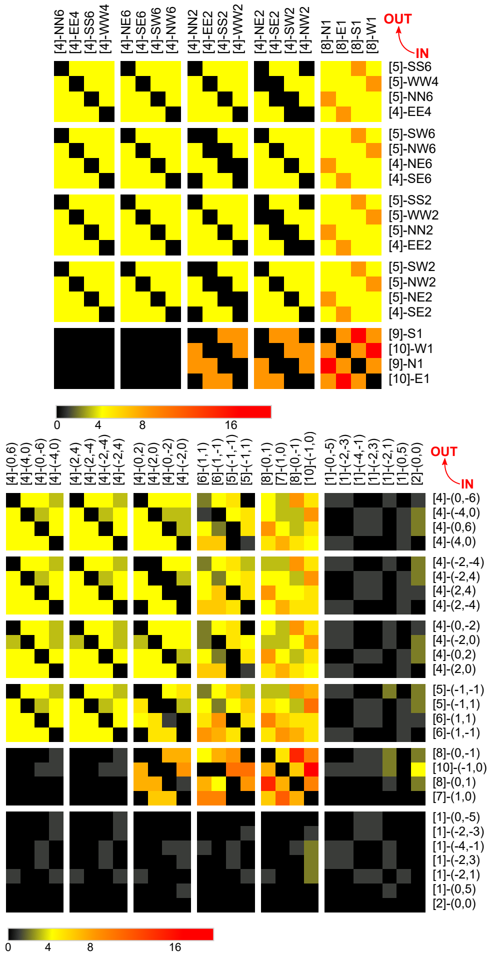
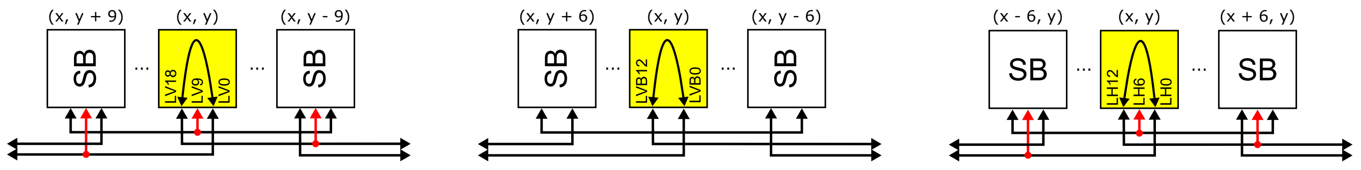


Figure 8: Short-range to mid-range adjacency analysis. Top, clustering by name; bottom, direction-vector clustering.

In the top part of Fig. 8, PIPJs are clustered by name. As such, the input sets sometimes have more members than the corresponding output sets, because of the way wire stubs (Figs. 6 and 7) are named. We see remarkably regular connectivity patterns, fully exposed only if stubs are not ignored. One could think that these regular patterns were, at some point in time, the actual design goal. In the bottom part of Fig. 8, direction-vector clustering is used (Section 4.2); this figure shows the *reality*, which, although slightly different, retains most of the regularity of the clustering by name. The difference could be a consequence of physical layout constraints—preventing or making the initial pattern too costly to implement—or it could have been intentionally introduced to create new routing opportunities at a small loss of regularity.

Immediately visible is that the number of wires each wire can drive (approximated by the sum of elements in one row over the corresponding input set size) vastly surpasses three, a value typically chosen in academia [46]. Focusing on the cardinal mid-range wires (LEN-2, LEN-4, LEN-6), we see that in almost all cases, they drive a LEN-1 wire of each type including those going in the direction from which the mid-range wire came, thus allowing for reaching



(a) Vertical LEN-18 wires (rotated by 90°). (b) Vertical LEN-12 wires (rotated by 90°). (c) Horizontal LEN-12 wires.

Figure 9: Topology of bidirectional long wires and the names of associated PIPJs. In red, input-only intermediate taps.

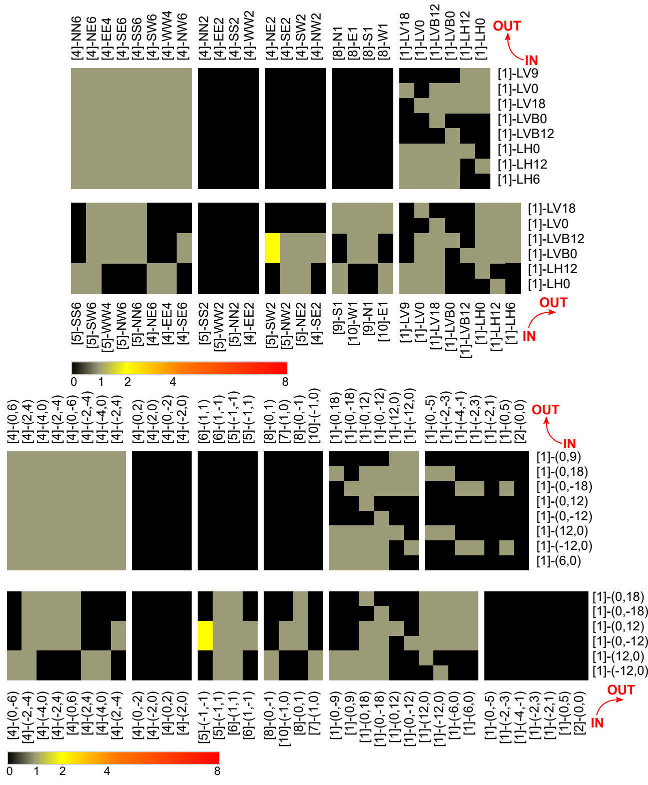


Figure 10: Long-range adjacency analysis. Top, clustering by name; bottom, direction-vector clustering.

intermediate CLBs with only one additional hop; this additionally improves routability in the absence of intermediate taps. Cardinal mid-range wires have access to almost twice as many LEN-1 wires in the same direction (e.g., four inputs (0, 2), arriving from south, can make eight connections to the four outputs (0, 1), going north), which favors routing straight connections as opposed to making bends. When it comes to connectivity with other mid-range cardinal wires, the usual principle of lack of connections to wires going in the direction from which the driving wire came, as this would result in unnecessary metal wastage, generally holds [36]. In particular, there are no connections to wires whose destination entirely coincides with the origin of the driving wire.

Diagonal wires (LEN-2, LEN-6) introduce a challenge in deciding which types of turns to support. We can observe that cardinal mid-range wires mostly do not drive LEN-2 diagonal wires that return to the half-plane of the driving wire source (with the boundary perpendicular to the direction of the driving wire). A possible reason for that could be that the CLBs which would be reachable by such connections are already reachable through the appropriate stubs of the LEN-1 wires. Looking at LEN-6 diagonal wires, all 90°-rotations of the E $\rightarrow$ SW connection exist, whereas those returning to the half-plane of the driver source are generally absent.

It is worth noting that the LEN-1 wires—likely routed at lower metal layers—very seldom drive LEN-4 and LEN-6 wires, which goes in hand with the conclusions of previous academic studies [53].

Let us now turn to long wires. Their respective connections are shown in Fig. 10. The labeling of long-wire PIPJs is explained in Fig. 9. We see that each long wire, including the intermediate input-only taps, drives one LEN-4 and one LEN-6 wire and none of the LEN-1 or LEN-2 wires. The LEN-12 horizontal and the LEN-18 vertical wires also drive all other long wires, whereas the LEN-12 vertical wire drives none of them. LEN-12 vertical wires also have more drivers coming from the short- and mid-range than any other long wire type. The combination of these two features probably makes them the fastest option for long distance vertical communication, because they allow increased access flexibility with lower capacitive loading, both due to the fewer sinks at the end and the lack of the intermediate tap. Perhaps the influence of such individually optimized connections on the critical path delay of the implemented designs is something that merits future investigation.

7.2 ALT and BOUNCE Connections

A signal carried by a global routing wire and arriving to the SB connected to the destination CLB, must pass through the PIPJ driving the target CLB input. In the absence of programmable connection between the incoming wire and the target PIPJ (e.g., distributed RAM data or control input) the signal may need to be rerouted to arrive from another global routing wire. To address this issue, the SBs in 7-Series FPGAs use a number of highly-interconnected PIPJs, called BYP-ALTs (BAs), FAN-ALTs (FAs), BYP-BOUNCEs (BBs), FAN-BOUNCEs (FBs), and GFANs.

Figs. 11, 12, and 13, show that BYP-ALTs and FAN-ALTs receive signals from LEN-1 and LEN-2 wires (never from the longs), from CLB outputs, and GFANs. Analyzing the connections further, we find that each BYP-ALT drives one of the BYP-BOUNCEs and one of the *bypass* inputs (AX in Fig. 1), whereas each FAN-ALT drives one of the FAN-BOUNCEs and one of the distributed RAM inputs.

BYP-BOUNCEs and FAN-BOUNCEs having fan-in of one suggests that those PIPJs are buffers or even simply labels for the short local connections. They pass their signal to at most two distinct LUT inputs (Fig. 14) or bounce it back to BYP-ALTs and FAN-ALTs, from where the path search to the desired CLB input can continue.

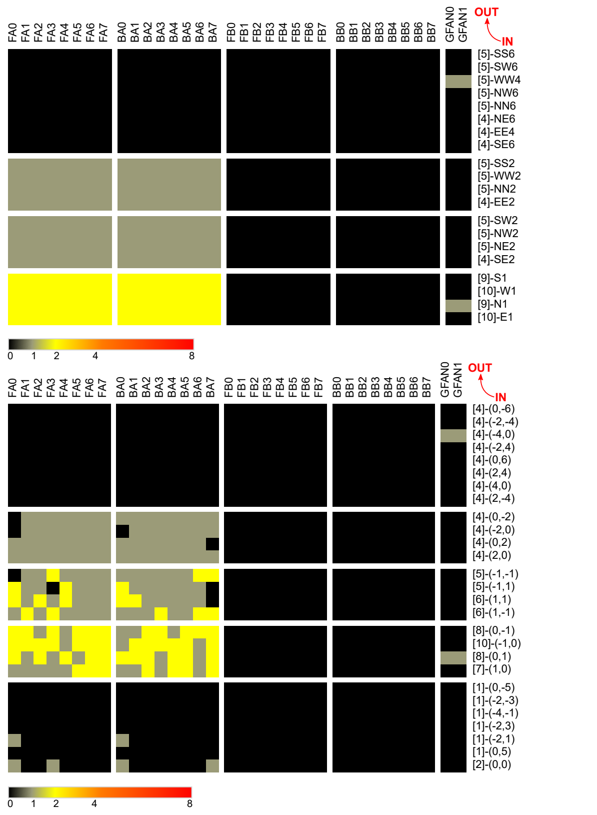


Figure 11: Adjacency of short- and mid-range wires and ALTs/BOUNCEs. Top, clustering by name; bottom, direction-vector clustering.

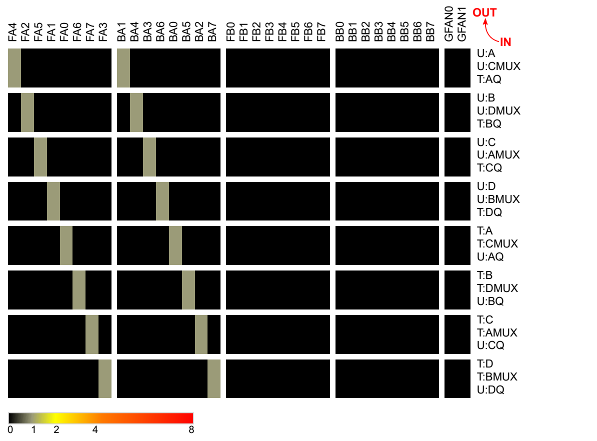


Figure 12: Adjacency of CLB outputs and ALTs/BOUNCEs.

GFANs can be driven by one of the BYP-BOUNCEs; perhaps more interestingly, they are the entry points for global clocks and the ground (GND in Fig. 3). As drivers, GFANs behave exactly like BYP-BOUNCEs and FAN-BOUNCEs.

Besides the above discussed PIPJs, we find additional eight SB inputs, labeled FAN-BOUNCE SOUTH and BYP-BOUNCE NORTH

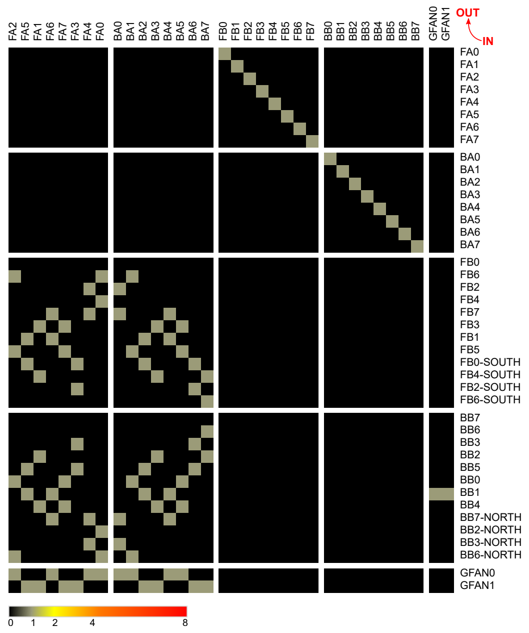


Figure 13: Adjacency of ALTs and BOUNCEs.

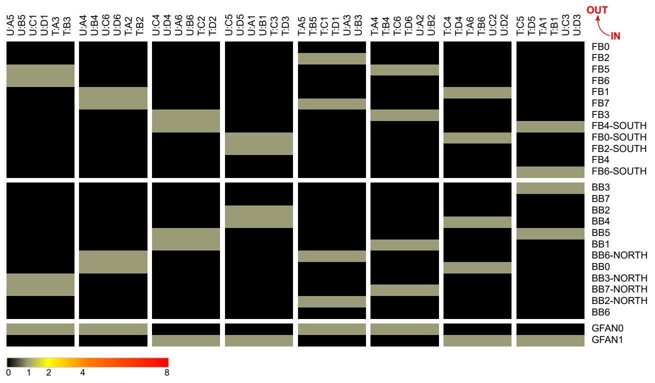


Figure 14: Adjacency of ALTs/BOUNCEs and LUT inputs.

in Fig. 15. Thanks to them, CLBs can drive the vertically-adjacent LUTs, without having to exit on a LEN-1 wire at all. Additionally, BYP-BOUNCE NORTH PIPJs open the way to the bypass inputs of the neighboring CLB.

7.3 CLB Connections

7.3.1 LUT Inputs. Fig. 16 shows that all 48 LUT inputs can be driven directly from the global routing, as has been indicated before [9, 23]. Therefore, there is no reduction of CLB input bandwidth typical of the academic and Intel architectures [7, 46]. Lifting this packing constraint potentially eases flat placement of LUTs using scalable techniques, such as analytical [18, 54], although it could prevent some area savings in the input multiplexing circuitry.

When clustering PIPJs by name, we find again a very regular pattern: each LUT input can be driven by one LEN-2 and two LEN-1 wires from each direction. The direction-vector clustering (bottom

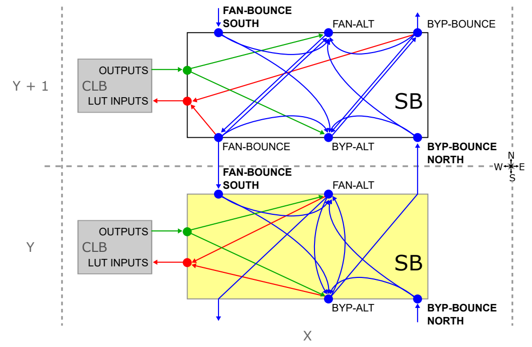


Figure 15: Vertically adjacent SBs can connect via FAN-BOUNCE NORTH and BYP-BOUNCE SOUTH. Top, full connectivity; bottom, equivalent and simplified connectivity, for FAN-BOUNCEs and BYP-BOUNCEs have fan-in of one.

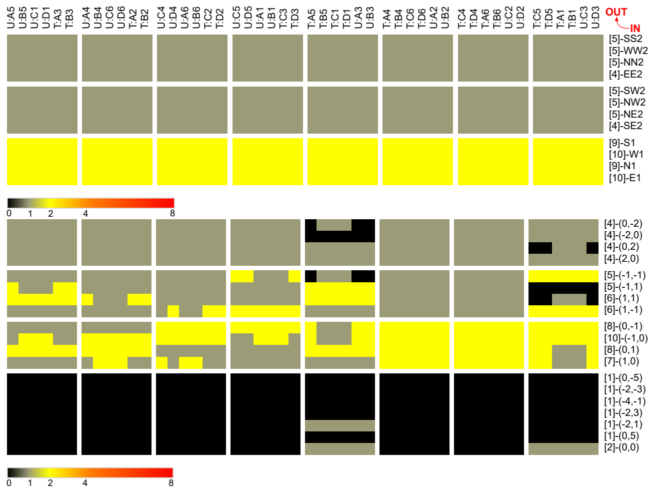


Figure 16: Adjacency of wires and LUT inputs. Top, clustering by name; bottom, direction-vector clustering. Wires longer than LEN-2 are omitted for they do not connect to CLB inputs.

of Fig. 16) once again breaks this regularity to an extent. Only LEN-1 and LEN-2 wires driving the LUT inputs goes in hand with the conclusions of the previous studies [53]. However, we discover that the possibility for some LUT inputs to be driven by LEN-4 or even LEN-6 wires is, in fact, given indirectly; this is achieved through the additional level of multiplexing, concealed behind `NL1BEG_N3` and `SR1BEG_S0`, the two special PIPJs mentioned in Section 6.1.

7.3.2 CLB Outputs. Connectivity between the CLB outputs and the routing wires is shown in Fig. 17. A typical output drives one mid- and two short-range wires of each type, reflecting their twice higher count. Long wires are generally not directly driven by the CLB, which is again in accordance with the previous conclusions [53]. This time again, we discover that some CLB outputs can drive LEN-18 vertical wires, albeit through an additional level of multiplexing, concealed behind `SR1BEG_S0` PIPJ (Section 6.1).

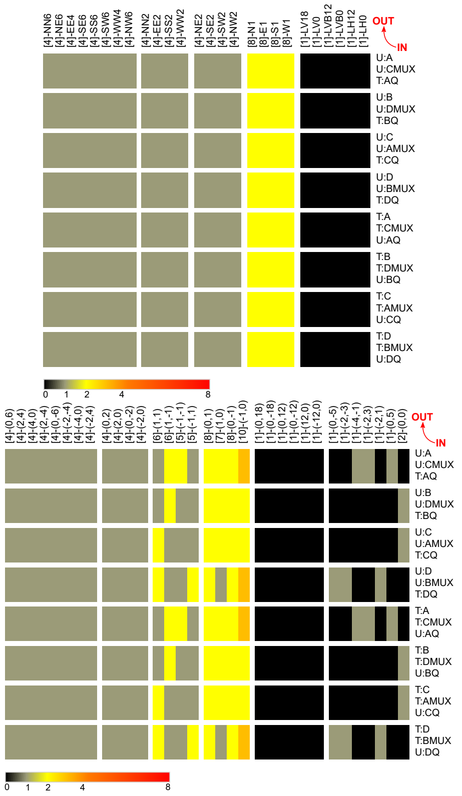


Figure 17: Adjacency of CLB outputs and wires. Top, clustering by name; bottom, direction-vector clustering.

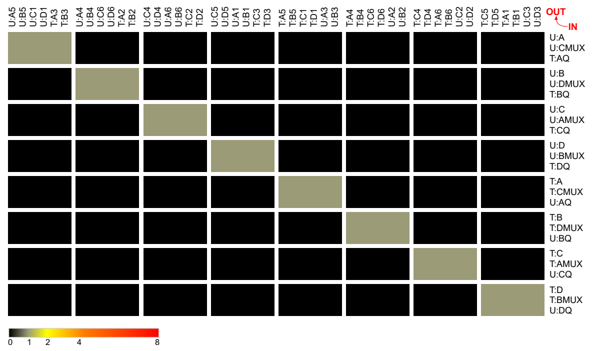


Figure 18: Adjacency of CLB outputs and LUT inputs.

Eight distinct equivalence classes of CLB outputs exist (Fig. 17). Although this number exactly matches the number of LUTs in the CLB and the type of the outputs in each class exactly matches those in each logic element (Fig. 1), we can see that the classes are not in fact formed by the outputs of individual logic elements, but rather come from entirely distinct ones. This could perhaps increase the flexibility of the connections between logic elements in different CLBs, in the absence of a crossbar. Permutations simply have to be considered closer to the source in that case, similarly to the new Agilex with the reduced input multiplexing capabilities [25]. Also potentially interesting is that the combinational outputs in each class come from the same slice (U or T), whereas the registered one comes from the other. This could reflect the physical distance of the driven wires from the actual location of the drivers and the fact that the registered output already cuts a timing path and is thus more likely to tolerate higher delay.

7.3.3 Feedback Connections. The feedback adjacency matrix is shown in Fig. 18. We can observe that the output equivalence classes from the previous section are preserved. Each LUT input is driven by all three elements of a single driver class. Inputs are themselves grouped into similar equivalence classes. Each O6 output can reach all LUTs in the same slice, directly or via the 6:1 multiplexer of the logic element (e.g., AMUX in Fig. 1), while it can reach half of the LUTs in the remaining slice through each of them; in other words, each 6-LUT can reach all LUTs in the entire CLB. In fractured mode, one of the 5-LUTs can again reach all LUTs in the entire CLB, while the other 5-LUT can reach all LUTs in the same slice and half the LUTs in the other. Although very sparse, this connectivity pattern allows almost arbitrary connectivity between LUTs in absence of congestion.

No two inputs of the same LUT are in the same class, meaning that although any LUT can reach any other LUT in the CLB through the feedback connections, it cannot reach just any desired input. Of course, when the inputs are logically equivalent, such input targeting may not even be necessary. Even if the inputs are logically equivalent, the typical academic model of an LUT containing a tree decoder [55] would assume that there is a large difference in the input delays, as is the case in the Intel architectures [56]. Having a possibility of connecting drivers to multiple LUT inputs of different speeds can be crucial for meeting the timing constraints in those circumstances. In 7-Series CLBs, this possibility does exist, via FAN-ALTs and BYP-ALTs. However, given the delay penalty of using multiple levels of multiplexing, we suspect that the alternative routes are rather targeting distributed RAM logic. Consequently, 7-Series LUTs may have a balanced input-delay profile, possibly owing to the full-CMOS-based decoder design with further optimizations towards such balancing [57]. This is another aspect that could change the usual assumptions entering the design of programmable interconnect and thus merits further investigation.

7.4 Multiplexers

So far, our discussion has focused on the various types of wires that exist in the 7-Series FPGAs and the switching possibilities between them. Multiplexers that actually perform the switching have been altogether neglected. In this section, we attempt to cast some light on them and how they might differ from the typical assumptions.

Let us start from the logic element internals. The two 6:1 muxes of Fig. 1 might seem a strange choice since a standard 2-level implementation of a 6:1 multiplexer would require 5 SRAM cells [58], whereas adding only one more would increase the number of available inputs to nine. It turns out, however, that the specificities of a 6:1 multiplexer allow it to be implemented using only three SRAM cells, still with two levels of steering switches [59]. The two muxes in Fig. 1 share five out of six inputs, which favors diffusion sharing and makes the combined layout even more compact [59].

Turning to the wire driver multiplexers of the SBs, they are all 20:1, which would again appear as an unintuitive choice in that it is close to but not quite a perfect square. Young again shows that a 20:1 multiplexer can be controlled by only five SRAM cells, instead of the typically assumed nine [58], albeit, this time with three levels of steering switches [59].

Finally, each LUT input can select from a set of 25 inputs, one of which is VDD. Due to the aforementioned efficiency of the 6:1 and 20:1 multiplexers, we trust that it is not a single 25:1 multiplexer driving an LUT input, but rather a 20:1 bringing in most of the external inputs, followed by a smaller 6:1 that provides fast propagation of three feedback signals (in Fig. 18). The remaining two inputs of the 6:1 multiplexers could act as fast inputs from the global routing, which is a number that has been previously suggested as sufficient [11]. Besides the feedbacks, each LUT input equivalence class (Section 7.3.3), shares 18 of the 22 external drivers, providing ample possibilities for pairing multiplexers in a way that would maximize diffusion sharing.

If our assumptions hold, this is likely another instance of layout efficiency being of prime importance during architectural design, which is something not often seen in academia where the majority of studies rely on simplistic transistor-counting models [55]. Not only would this change the assumptions on area cost of multiplexers, but it could also change the assumptions on the delay penalties they induce, due to the existence of three steering switches on paths through most of them. Finally, although wider multiplexers are penalized in a typical academic architectural study, to the best of our knowledge, it was never attempted to tailor the detailed routing architecture towards a particular, efficient multiplexer size—it was rather the multiplexer size that came as a result of a particular detailed architecture design. A similar statement could be made about maximizing input sharing between adjacent multiplexers. We are not aware of that ever being an actual optimization goal. In fact, even the area models which account for impact of diffusion sharing between transistors within the same multiplexer do not consider what is the impact of diffusion sharing between pairs of multiplexers [60]. Perhaps an even more important factor today that could stimulate input sharing as an objective, is an increased possibility to share vias [25].

8 CONCLUSION

In this work, we presented NetCracker—a flexible framework for extracting the characteristics of FPGA routing architectures and raising them to a level of abstraction that facilitates general understanding. Our conjectures are that, in order to stand up to the challenges of technology scaling and ever more demanding acceleration tasks, the FPGA routing architecture will have to evolve and that the broader research community could be of great use in making this evolution successful. The main premise on which this work was based is that understanding existing routing architectures is necessary for providing the right basis for innovation. We believe that NetCracker is timely in making this learning process easier and more efficient, which we tried to demonstrate on the previously unexplored 7-Series architecture family of Xilinx. Our hope is that the conclusions presented here already make a small first step towards narrowing the gap between academic research and industrial reality—a gap which has been considerably widened in the last few years.

REFERENCES

[1] J. Rose, A. E. Gamal, and A. Sangiovanni-Vincentelli, "Architecture of field-programmable gate arrays," *Proceedings of the IEEE*, vol. 81, no. 7, pp. 1013–29, Jul. 1993.

[2] S. Singh, J. Rose, P. Chow, and D. Lewis, "The effect of logic block architecture on FPGA performance," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 3, pp. 281–87, Mar. 1992.

[3] D. Lewis and J. Chromczak, "Process technology implications for FPGAs (invited paper)," in *2012 International Electron Devices Meeting*, San Francisco, CA, USA, Dec. 2012, pp. 25.2.1–4.

[4] B. Gaide, D. Gaitonde, C. Ravishankar, and T. Bauer, "Xilinx adaptive compute acceleration platform: Versal architecture," in *Proceedings of the 27th ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, Seaside, CA, USA, Feb. 2019, pp. 84–93.

[5] R. Nijssen, "FPGAs will never be the same again: How the newest FPGA architectures are totally disrupting the entire FPGA ecosystem as we know it," in *Proceedings of the 28th ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, Seaside, CA, USA, Feb. 2020, p. 172.

[6] "Understanding peak floating-point performance claims (white paper)," Available: www.intel.com, Intel Corporation, 2017.

[7] 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 11th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, Monterey, CA, USA, Feb. 2003, pp. 12–20.

[8] D. Lewis, G. Chiu, J. Chromczak, D. Galloway, B. Gamsa, V. Manohararajah, I. Milton, T. Vanderhoek, and J. V. Dyken, "The Stratix™ 10 highly pipelined FPGA architecture," in *Proceedings of the 24th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, Monterey, CA, USA, Feb. 2016, pp. 159–68.

[9] E. Hung, F. Eslami, and S. J. E. Wilton, "Escaping the academic sandbox: Realizing VPR circuits on Xilinx devices," in *Proceedings of the 21st IEEE Symposium on Field-Programmable Custom Computing Machines*, Seattle, WA, USA, Apr. 2013, pp. 45–52.

[10] J. Luu, "Architecture-aware packing and CAD infrastructure for field-programmable gate arrays," Ph.D. dissertation, University of Toronto, 2014.

[11] D. Lewis, E. Ahmed, G. Baeckler, V. Betz, M. Bourgeault, D. Cashman, D. Galloway, M. Hutton, C. Lane, A. Lee, P. Leventis, S. Marquardt, C. McClintock, K. Padalia, B. Pedersen, G. Powell, B. Ratchev, S. Reddy, J. Schleicher, K. Stevens, R. Yuan, R. Cliff, and J. Rose, "The Stratix II logic and routing architecture," in *Proceedings of the 13th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, Monterey, CA, USA, Feb. 2005, pp. 14–20.

[12] D. Lewis, E. Ahmed, D. Cashman, T. Vanderhoek, C. Lane, A. Lee, and P. Pan, "Architectural enhancements in Stratix-III™ and Stratix-IV™," in *Proceedings of the 17th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, Monterey, CA, USA, Feb. 2009, pp. 33–42.

[13] 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 21st ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, Monterey, CA, USA, Feb. 2013, pp. 147–56.

[14] V. Betz and J. Rose, "VPR: A new packing, placement and routing tool for FPGA research," in *Proceedings of the 7th International Conference on Field-Programmable Logic and Applications*, London, UK, Sep. 1997, pp. 213–22.

[15] Z. Li, Y. Xiao, Y. Zhang, Y. Pang, C. Hu, J. Wang, and J. Lai, "An automatic transistor-level tool for GRM FPGA interconnect circuits optimization," in *Proceedings of the 2019 on Great Lakes Symposium on VLSI*, Tysons Corner, VA, USA, May 2019, pp. 93–98.

[16] Z. Jiang, C. Y. Lin, L. Yang, F. Wang, and H. Yang, "Exploring architecture parameters for dual-output LUT-based FPGAs," in *Proceedings of the 24th International Conference on Field-Programmable Logic and Applications*, Munich, Germany, Sep. 2014, pp. 1–6.

[17] (2020) Symbiflow [Online]. Available: symbiflow.github.io/.

[18] T. Ahmed, P. D. Kundarewich, and J. H. Anderson, "Packing techniques for Virtex-5 FPGAs," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 2, no. 3, pp. 18:1–24, Sep. 2009.

[19] H. Fraisse, A. Joshi, D. Gaitonde, and A. Kaviani, "Boolean satisfiability-based routing and its application to Xilinx UltraScale clock network," in *Proceedings of the 24th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, Monterey, CA, USA, Feb. 2016, pp. 74–79.

[20] H. Fraisse and D. Gaitonde, "A SAT-based timing driven place and route flow for critical soft IP," in *Proceedings of the 28th International Conference on Field-Programmable Logic and Applications*, Dublin, Ireland, Aug. 2018, pp. 8–15.

[21] S. Yang, A. Gayasen, C. Mulpuri, S. Reddy, and R. Aggarwal, "Routability-driven FPGA placement contest," in *Proceedings of the 2016 on International Symposium on Physical Design*, Santa Rosa, CA, USA, Apr. 2016, pp. 139–43.

[22] S. Yang, C. Mulpuri, S. Reddy, M. Kalase, S. Dasasathyan, M. E. Dehkordi, M. Tom, and R. Aggarwal, "Clock-aware FPGA placement contest," in *Proceedings of the 2017 ACM on International Symposium on Physical Design*, Portland, OR, USA, Mar. 2017, pp. 159–64.

[23] S. Chandrakar, D. Gaitonde, and T. Bauer, "Enhancements in UltraScale CLB architecture," in *Proceedings of the 23rd ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, Monterey, CA, USA, Feb. 2015, pp. 108–16.

[24] J. Tyhach, M. Hutton, S. Atsatt, A. Rahman, B. Vest, D. Lewis, M. Langhammer, S. Shumarayev, T. Hoang, A. Chan, D.-M. Choi, D. Oh, H.-C. Lee, J. Chui, K. C. Sia, E. Kok, W.-Y. Koay, and B.-J. Ang, "Arria™ 10 device architecture," in *Proceedings of the IEEE Custom Integrated Circuit Conference*, San Jose, CA, USA, May 2015, pp. 1–8.

[25] 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 28th ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, Seaside, CA, USA, Feb. 2020, pp. 140–49.

[26] Xilinx Inc., *7 Series FPGAs Configurable Logic Block User Guide (UG474)*, Sep. 2016.

[27] ——, *Vivado Design Suite Tcl Command Reference Guide (UG835)*, May 2019.

[28] C. Wolf and M. Lasser. (2020) Project IceStorm. Available: http://www.clifford.at/icestorm/.

[29] N. Steiner, A. Wood, H. Shojaei, J. Couch, P. Athanas, and M. French, "Torc: Towards an open-source tool flow," in *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, Monterey, CA, USA, Feb. 2011, pp. 41–44.

[30] C. Lavin, M. Padilla, J. Lamprecht, P. Lundrigan, B. Nelson, and B. Hutchings, "RapidSmith: Do-it-yourself CAD tools for Xilinx FPGAs," in *Proceedings of the 21st International Conference on Field-Programmable Logic and Applications*, Chania, Greece, Sep. 2011, pp. 349–55.

[31] (2020) Project X-ray [Online]. Available: github.com/SymbiFlow/prjxray/.

[32] (2020) Project Trelis [Online]. Available: github.com/SymbiFlow/prjtrellis/.

[33] 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 (TRETS)*, vol. 13, no. 2, pp. 9:1–9:60, May 2020.

[34] B. F. Fawcett, "User-programmable gate arrays: Design methodology and development systems," *Microprocessors and Microsystems*, vol. 13, no. 5, pp. 321–27, Jun. 1989.

[35] Xilinx Inc., *The Programmable Gate Array Data Book*, 1989.

[36] 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. 277–82, Mar. 1991.

[37] V. Betz and J. Rose, "FPGA routing architecture: Segmentation and buffering to optimize speed and density," in *Proceedings of the 7th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, Monterey, CA, USA, Feb. 1999, pp. 59–68.

[38] ——, "How much logic should go in an FPGA logic block," *IEEE Design and Test of Computers*, vol. 15, no. 1, pp. 10–15, Jan. 1998.

[39] A. DeHon, "Balancing interconnect and computation in a reconfigurable computing array (or, why you don't really want 100% LUT utilization)," in *Proceedings of the 7th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, Monterey, CA, USA, Feb. 1999, pp. 69–78.

[40] (2020) NetCracker: Programmable routing architecture analysis framework. Available: github.com/mortbopet/netcracker.

[41] Xilinx Inc., *7 Series FPGAs Data Sheet (ds180)*, Sep. 2020.

[42] Z. Seifoori, H. Asadi, and M. Stojilović, "A machine learning approach for power gating the FPGA routing network," in *Proceedings of the IEEE International Conference on Field Programmable Technology*, Tianjin, China, Dec. 2019, pp. 10–18.

[43] V. Betz and J. Rose, "Directional bias and non-uniformity in FPGA global routing architectures," in *Proceedings of the International Conference on Computer Aided Design*, San Jose, CA, USA, Nov. 1997, pp. 652–59.

[44] C. Mead and L. Conway, *Introduction to VLSI Systems.* Reading, Mass.: Addison-Wesley, 1980.

[45] M. Hutton, "Characterization and parameterized generation of digital circuits," Ph.D. dissertation, University of Toronto, 1997.

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

[47] G. Lemieux, E. Lee, M. Tom, and A. Yu, "Directional and single-driver wires in FPGA interconnect," in *Proceedings of the IEEE International Conference on Field Programmable Technology*, Brisbane, NSW, Australia, Dec. 2004, pp. 41–48.

[48] *Virtex-II Platform FPGA Handbook*, Xilinx Inc., 2001.

[49] S. Sivaswamy, G. Wang, C. Ababei, K. Bazargan, R. Kastner, and E. Bozorgzadeh, "HARP: Hard-wired routing pattern FPGAs," in *Proceedings of the 13th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, Monterey, CA, USA, Feb. 2005, pp. 21–29.

[50] X. Sun, H. Zhou, and L. Wang, "Bent routing pattern for FPGAs," in *Proceedings of the 29th International Conference on Field-Programmable Logic and Applications*, Barcelona, Spain, Sep. 2019, pp. 9–16.

[51] A. Roopchansingh and J. Rose, "Nearest neighbour interconnect architecture in deep submicron FPGAs," in *Proceedings of the IEEE Custom Integrated Circuit Conference*, Orlando, FL, USA, May 2002, pp. 59–62.

[52] J. Gould. (2020) Morpheus. Available: software.broadinstitute.org/morpheus.

[53] O. Petelin and V. Betz, "The speed of diversity: Exploring complex FPGA routing topologies for the global metal layer," in *Proceedings of the 26th International Conference on Field-Programmable Logic and Applications*, Lausanne, Switzerland, 2016, pp. 1–10.

[54] W. Li and D. Z. Pan, "A new paradigm for FPGA placement without explicit packing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 11, pp. 2113–26, Nov. 2019.

[55] C. Chiasson and V. Betz, "COFFE: Fully-automated transistor sizing for FPGAs," in *Proceedings of the IEEE International Conference on Field Programmable Technology*, Kyoto, Japan, Dec. 2013, pp. 34–41.

[56] K. E. Murray, S. Whitty, S. Liu, J. Luu, and V. Betz, "Timing-driven Titan: Enabling large benchmarks and exploring the gap between academic and commercial CAD," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 8, no. 2, pp. 10:1–18, Mar. 2015.

[57] M. Chirania, "Lookup table with relatively balanced delays," US Patent 7471,104 B1, 2008.

[58] D. Koch, *Partial Reconfiguration on FPGAs*, ser. Lecture Notes in Electrical Engineering. Springer-Verlag New York, 2013, vol. 153.

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

[60] F. F. Khan, "Towards accurate FPGA area models for FPGA architecture evaluation," Ph.D. dissertation, Ryerson University, 2017.