IIBLAST: Speeding Up Commercial FPGA Routing by Decoupling and Mitigating the Intra-CLB Bottleneck

Shashwat Shrivastava*, Stefan Nikolić*, Chirag Ravishankar†, Dinesh Gaitonde†, and Mirjana Stojilović*
*EPFL, †AMD
{shashwat.shrivastava, stefan.nikolic, mirjana.stojilovic}@epfl.ch, {chirag.ravishankar, dinesh.gaitonde}@amd.com

***Abstract*—We identified that in modern commercial FPGAs, routing signals from the general interconnect to the configurable logic blocks (CLBs) through a very sparse input interconnect block (IIB) represents a significant runtime bottleneck. This is despite academic research usually altogether neglecting the runtime of *last-mile* routing through the IIB. To alleviate this bottleneck, we combine computer-aided design (CAD) and FPGA architecture enhancements. We propose a multi-stage FPGA routing approach, based on the premise that once the signals are legally routed in general interconnect—only reaching the inputs of the IIB, but not the final targets—the remaining last-mile routing through the IIB can be completed efficiently and independently for each FPGA tile. Then, the final routing solution can simply be built by joining the previously obtained partial solutions. However, we observe that some properties of modern IIB architectures limit the success rate of the intra-CLB routing, creating the need for revisiting the routing in the general interconnect and inevitably impairing the multi-stage routing runtime gains. We show that an enhanced IIB architecture mitigates the issue at a minimal cost.**

With ISPD16 benchmarks and an FPGA architecture model closely resembling AMD UltraScale FPGAs, we demonstrate the dominant contribution of last-mile routing to the router's runtime. After applying our multi-stage routing approach and the proposed enhancements, we show that the observed bottleneck can be mitigated, resulting in 4.94× faster routing on average.

***Index Terms*—CLB, congestion, FPGA, IIB, routing**

I. Introduction

It is well known that routing, besides placement, takes up an important part of the *field programmable gate array* (FPGA) *computer-aided design* (CAD) flow runtime [1], [2]. Although there are a number of published attempts at parallelizing routing [3], for highly congested designs—which are the truly relevant ones as they drive the runtime up—there are too many conflicting nets preventing the common workload-partitioning methods from being effective. On the other hand, FPGA placement is much more amenable to parallelization [4], and it is only expected that the share of the runtime for routing will keep increasing. That is unless something is radically changed in routing, which is precisely what this paper attempts.

The main premise behind this work is that routing itself has a major runtime bottleneck created by the way in which commercial FPGA architectures are designed. Before experimentally identifying this bottleneck, in this section, we back up our expectations by analyzing qualitatively the main aspects of these architectures and how they impact the routing process.

A. Highways and City Streets: Organization of FPGA Interconnect

FPGA routing architecture can conceptually be divided into two main parts: 1) *general interconnect*, which steers a signal from its source to its target logic cluster (CLB) and 2) *input interconnect block* (IIB), which dispatches it to its final destination within the logic cluster. This is illustrated in Fig. 1 and will be discussed in more detail in Section II-A. Very loosely, we could consider the general interconnect and the IIB as analogous to a highway network and a city street network, respectively: although cars traveling between two cities spend the majority of their time on highways, each of them has to pass through a set of city streets to reach the destination address. The IIB is usually organized as a two-level multiplexing structure like the one shown in Fig. 2.

This work is partially supported by the Swiss National Science Foundation (grant No. 182428).

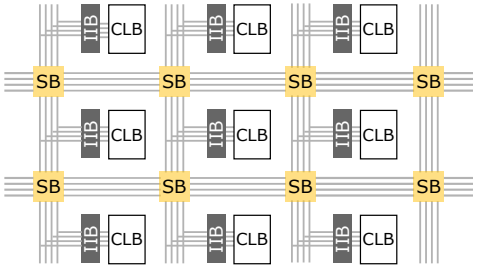


Fig. 1: High-level view of the CLBs and the FPGA interconnect. Routing multiplexers are organized in two groups: input interconnect blocks (IIBs) and switch blocks (SBs).

B. A 10,000 Foot View of FPGA Routers

We will review the principles of *congestion negotiation* [5]—which most modern FPGA routers are based on [6]—in more detail in Section II-B. For now, it is sufficient to note that signals are routed iteratively, one at a time, each using the shortest path through the available routing resources. Whenever a signal is traced through a particular resource u, u's cost is increased so that the other signals are incentivized to avoid it. However, it may still happen that two signals overlap on any given resource, which creates *congestion* [7]. In case of congestion, the overlapping signals must be *ripped up* and *rerouted*, in the hope that this time the conflict will be avoided [5].

Two main factors impact the runtime of an FPGA router: 1) the number of times each signal has to be rerouted because it overlapped with another and 2) the number of resource expansions that the shortest-path algorithm needs to make while seeking the path through the programmable interconnect that will connect the signal's source and destination (sink) endpoints, fixed during placement.

C. Do Not Plan the Streets Before the City is Reached

Returning to the loose road network analogy, routing a circuit on an FPGA could be considered analogous to planning routes for a multitude of cars in a centralized manner, such that traffic jams are prevented in every highway section and every city street. Most of the time, when a congestion-free route plan has been created for the highway network, completing the plan within the city street network will be possible without reconsidering the distribution of cars among the highway exits, which would require updates to the highway network plan. In other words, routing through the highway

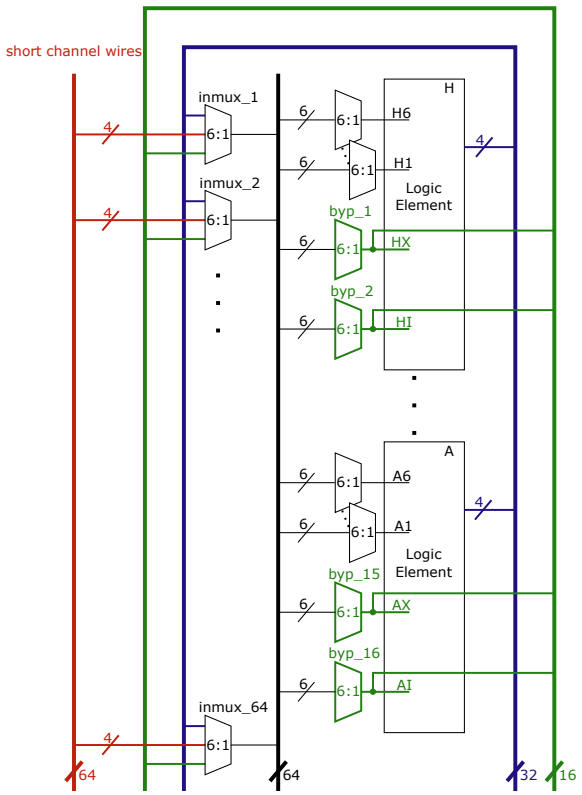


Fig. 2: Detailed IIB. In blue, feedback connections. In green, bypass pin connections. In red, channel wires spanning one or two tiles.

and the city street networks can be planned largely independently. This is the main idea which we build upon in this work.

However, standard commercial FPGA routers do not apply this split, instead they route each signal until the target pin, every time it is ripped up. To reach the target pin within the target CLB, the signal has to pass through an IIB. Each time there is an overlap in the general interconnect and a signal has to avoid a resource that it used previously, the downstream portion of the path has to be rerouted as well; at the very least, that includes routing through the IIB. Already this hints at the possibility of intra-CLB routing being the runtime bottleneck. Unfortunately, problems with intra-CLB routing do not stop with the frequency at which it is required. To limit the number of expansions while finding the shortest paths, FPGA routers usually rely on A*, with the heuristic being related to the minimum distance from the current to the target cluster [1]. This is very effective for the path's portion through the general interconnect, but once the signal reaches the target IIB, any heuristic based on minimum cluster distance is of no further use—inside the IIB, the shortest path algorithm reduces to Dijkstra.

Considering that the IIB has a depth of two and taking into account the number and small size of multiplexers shown in Fig. 2, representative of AMD UltraScale FPGAs [8], we can conclude that the absence of A* inside the IIB is unlikely to be a major concern. As will be shown in Section VI, when perceived in isolation, it is not. However, when coupled with the frequency at which IIB routing is typically required, the absence of A* is strongly felt.

The same small multiplexers of commercial IIBs that help node expansion reduction while routing with no or ineffective A* complicate the issue regarding reducing the number of times each signal has to be ripped up. Namely, the limited number of routing possibilities through a sparse commercial IIB increases the likelihood of congestion, causing more rip-ups, often in the general interconnect as well.

D. Exploiting the Analogy

In this paper, we build on the intuition developed in this section and, in Section V, demonstrate experimentally that repeated IIB rerouting is a major runtime bottleneck. Then, in Section VI, we propose a new multi-stage routing algorithm that leverages the highway and city street analogy we introduced above to resolve the identified bottleneck. Essentially, we take the approach of early versions of the *Versatile Place and Route* (VPR) academic FPGA router, developed at the time when IIB architectures were fully connected [7]: routing all signals *only* until the IIB boundary, knowing that routing through the IIB itself could always be done because the structure is fully connected. For modern, very sparse IIB architectures such as the one shown in Fig. 2, we, of course, cannot *know* that all signals could be routed until their target CLB pins without creating overlaps, for any combination of IIB pins through which they enter the cluster. However, in this work, we *assume* that *almost* always, the sparse IIB would offer enough flexibility to complete the routing until the target pins without having to reroute the paths in the general interconnect. This is very similar to assuming that city street networks are usually flexible enough to complete a congestion-free last-mile routing for any reasonable distribution of cars among highway exits leading to the city. That assumption allows us to completely ignore IIB routing in *Step 1* of the proposed multi-stage algorithm, much like VPR used to do. But, as the sparse commercial IIBs are not guaranteed to be able to complete the routing, in *Step 2* we route independently each intra-CLB routing problem as well. Finally, given the IIB's sparsity, for some CLBs, a legal routing solution is inevitably impossible to find. After composing all subproblems into a single partially legal solution, we fully legalize it by incremental rerouting in *Step 3*.

The extent to which our assumption that most IIBs can be routed independently in practice actually holds is reflected on the time taken for this final incremental rerouting. After presenting the results of applying the proposed multi-stage routing algorithm (Section VI), and analyzing some causes of assumption violation, we suggest a relatively low-cost architecture enhancement that greatly increases the effectiveness of the algorithm (Section VII). Finally, after discussing the relation of our contribution to prior work in Section VIII, we conclude in Section IX.

II. Preliminaries

Before quantifying the bottleneck that routing through the commercial sparse IIBs presents, we need to build a model of the representative commercial FPGA architecture, including the realistic IIB, and set up the software environment to run the routing. As common in academic research on FPGA architecture and CAD, we use the open-source *Verilog-to-Routing* (VTR) tool flow [1], which includes VPR for placement and routing. In this section, we describe the FPGA architecture modeling and the VPR implementation of the PathFinder negotiated routing algorithm.

A. FPGA Architecture and Modeling

We explore FPGA architectures closely resembling those of the AMD UltraScale family [8]. *Island-Style* FPGAs [9], including UltraScale, are constructed by abutting a number of identical *tiles*, as was illustrated in Fig. 1. The typical FPGA tile consists of a *Configurable Logic Block* (CLB), wires grouped in horizontal and vertical routing channels, a large number of routing multiplexers split into the IIB, which has already been presented to some extent in Section I, and a

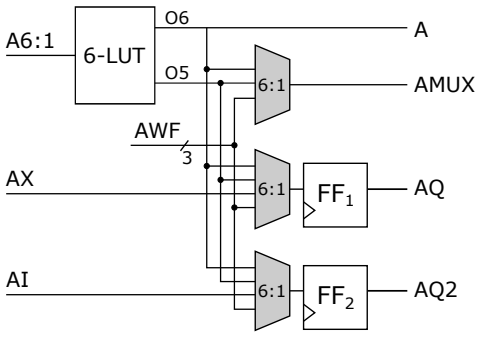


Fig. 3: Single logic element of a CLB in the UltraScale FPGA architecture. The bypass inputs AX and AI provide direct connections between the general interconnect and the flops. Carry and wide-function logic, which drive the remaining inputs (AWF) of the three multiplexers [11], are not shown.

switch block (SB), which contains all of the multiplexers that drive the channel wires, taking inputs both from other channel wires and the CLB itself.

1) General Interconnect: All FPGA architectures that we will model in this paper share a common general interconnect architecture [8]; that is, their channel wires and SB multiplexers are the same. Wires come in lengths one, two, four, and twelve (determined by the number of SBs they span). Each tile contains eight instances of a wire in each length-direction. CLB inputs are driven by length one and two wires only. A detailed description of the general routing architecture is available in our artifact repository [10].

2) Logic Elements: In the UltraScale architecture, a CLB contains eight *logic elements* (LEs), labeled A–H in Fig. 2. The main components of an LE, as shown in Fig. 3 for logic element A, are one 6-input dual-output *Look-Up Table* (LUT), two flip-flops (FFs), and several multiplexers. Apart from the six inputs of the LUT, each LE has two additional *bypass* (BYP) inputs (X and I), allowing the FFs to be driven independently of the LUT. This arrangement benefits highly-pipelined designs and permits better utilization of the resources [8]. Therefore, the city streets in our analogy need to be able to service 64 destinations.

3) IIB Internals: Let us now briefly return to the IIB of Fig. 2. It is dimensioned to receive signals from 64 wires in the global interconnect and to route them to 64 destinations inside the CLB. Indicated in green are the second-layer multiplexers that feed the bypass inputs. What differentiates them from the other second-layer multiplexers is that their outputs are fed back to the first layer; each first-layer multiplexer receives one input from a bypass multiplexer. This allows signals to enter the cluster from the general interconnect through some first-layer multiplexer, from which the target pin is not directly reachable, but eventually reach the target through a (number of) bypass multiplexer(s). In other words, once cascading through the bypass multiplexers is considered, all channel wires can reach any LUT input. In this way, the connectivity of the commercial IIBs is greatly extended without any increase in the number and size of its constituent multiplexers. However, it must be noted that the number of bypass multiplexers is fairly small, meaning that if many signals enter the CLB through a first-layer multiplexer far from the final target, bypass multiplexers can quickly become congested. To make the situation even more difficult, whenever the FFs are used independently, corresponding bypass multiplexers become unavailable for first-layer multiplexer switching; at that point, the only remaining solution is for the router to reroute the signals further upstream in the general interconnect. Unfortunately, in modern, highly pipelined designs, it is rather common for the FFs to be used independently [8], [12].

To construct the IIBs and SBs which conform to the above descriptions and that we can make openly available, we use the algorithm of Lemieux and Lewis [13], which also allows us to sweep the space of architectural choices and evaluate their impact. Even though the connectivity patterns in real devices are slightly more constrained because of specific wirelength requirements and physical limitations, the metrics of interest for this work are comparable between the modeled and real architectures. In the remainder of the paper, we keep our focus on the modeled architectures.

4) Routing Architecture Modeling: In VPR, the FPGA interconnect architecture is modeled as a directed *Routing-Resource Graph* (RRG), in which channel wires, primitive pins (e.g., LUT inputs and outputs), and the outputs of first-layer multiplexers in the IIB are represented as nodes. Additional *virtual* nodes represent the swappability of LUT input pins [7], [14]; in our model, for simplicity, we consider all six inputs of the dual-output LUT to be permutable. Edges model programmable connections between the nodes.

In VPR, IIB-like structures are usually described by listing the appropriate multiplexers within the CLB itself, so that they are only visible to the packer and not to the router [15], thus excluding routing through IIB. We circumvent this by exposing the IIB outputs to the general interconnect and representing the IIB multiplexers as separate nodes in the RRG, like Moctar et al. suggest [16]. The same approach has been used by the SymbiFlow project, which provides support for implementing designs on AMD 7-Series FPGAs using VPR [17].

B. Negotiated Congestion Routing

Similarly to commercial FPGA routers [6], VPR router is based on PathFinder negotiated congestion algorithm [5], which iteratively finds the shortest paths in the RRG for all source-sink pairs in the routing problem, one at a time. Each shortest path is sought using an A^* algorithm. Each node is associated with a cost and while routing, a heap structure is employed to keep the relevant nodes sorted by their cost. While searching for the shortest path, the router iteratively pops the cheapest node from the heap, then pushes all of its unvisited children onto the heap until the target is reached [1]. Since each signal is routed in this greedy fashion, overlaps between signals can occur; overlaps represent *congestion* and constitute a legality violation. The cost of congested nodes is then gradually increased, in proportion with the number of signals that use the given node [5]; as a result, the subsequently routed signals are redirected toward unused nodes. Once all of the signals are routed, at least those which use congested nodes are ripped up [1] and routed again in the next iteration. Between two routing iterations, the cost of congested nodes is further increased to drive the algorithm towards converging to a legal solution.

The router runtime is clearly determined by two effects: the number of shortest paths that need to be found (including those that must be ripped up and rerouted), which is directly related to the amount of congestion present in the RRG, and the number of RRG expansions required to find each shortest path. The structure of the IIB plays an important role here: sparser connectivity can increase the number of nodes that have to be expanded to reach the target, and it can also lead to fewer possibilities to route each signal, potentially increasing the congestion and driving the rip-up count up. On the other hand, higher fanouts increase the number of nodes that must be pushed onto the heap on each expansion. To measure the architecture impact on the router performance, we will report the runtime and the heap operation count (the latter being a machine-agnostic metric).

TABLE I: Characteristics of ISPD16 contest benchmarks [18], [21].

	LUT	FF	RAM	DSP
Min	50K	55K	0	0
Max	500K	602K	1000	600
Average	327.5K	291K	675	450

III. Experimental Setup

All the experiments described in the subsequent sections use the experimental setup described here. The experiments are run on an Ubuntu 22.04 LTS machine equipped with Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz (24 cores, 48 threads) and 256GB RAM. Using the VPR router (commit No. 33c518fc6) as a foundation, we provide the artifacts to reproduce the paper's results [10]. As benchmarks, we use openly-available circuits from the ISPD16 design contest on routability-driven FPGA placement (by Xilinx/AMD) [18], their corresponding UTPlaceF placements [19], and the UltraScale architectural model (with FPGA size 168×480) from the VTR repository. Table I highlights the characteristics of the benchmarks.

To import the placements, we first pack all LUTs and FFs placed in the same CLB into a single CLB primitive. This netlist is converted from the bookshelf to the blif format using the script provided in the VTR repository [1]. VTR can then convert the packing into the required format by trivially rewriting it (each primitive in the packed netlist can be converted into exactly one block by VTR). Finally, we convert the placement provided by UTPlaceF to the required format, making it compatible with the VTR-rewritten packing.

The UltraScale architectural model in the VTR repository is intended for placement and, consequently, lacks a realistic model of the interconnect architecture (the connectivity and delay information are not completely and appropriately specified). To overcome this issue, we include the interconnect model described in Section II-A through an external file describing the corresponding RRG, much like it is done by the SymbiFlow project, for example [17].

Lastly, being intended for routability-driven placement, ISPD16 benchmarks lack timing information. They are examples of difficult-to-route designs for which the routing runtime and wirelength are of higher concern than the critical path delay. Many practical FPGA applications fall in this category, notably ASIC emulation and FPGA hardware prototyping [20], for which routing runtime is one of the key factors determining the time to market. The experiments presented in this paper will, therefore, thoroughly evaluate how suitable our novel routing approach is for large and challenging to route, but not timing-critical designs.

IV. Enhancing the Baseline

To ensure a comprehensive evaluation of the runtime improvements discussed in this paper, a fair baseline is required. Through experimentation, we have found that adjusting some of the VPR router parameters reduces its runtime. This section explains our findings.

During routing, nodes of the RRG are assigned costs which help resolve congestion by increasing the cost of congested nodes and determining the shortest path by considering lower-cost nodes. In VPR, the cost is a function of present and historical congestion; the former is updated after a signal is routed, while the latter is updated between two subsequent routing iterations. The overall cost of a node, which determines where the node will be inserted in the ordered heap, depends not only on the two previously mentioned factors but also on a *router lookahead*—that is, the A* heuristic. The lookahead estimates the cost of a path from a node until the target pin. The node cost, the lookahead estimate, and the actual cost of the path from the source pin to the node are accumulated to compute the total cost of the node when inserting it in the heap [1]:

$$\begin{aligned} TotalCost(u) = & PathCost(source, u) + \\ & \alpha(u) \cdot pres\_fac \cdot (1 + NumSignals(u)) + \\ & Lookahead(u, target). \end{aligned}$$

Here, we use $\alpha(u)$ to designate the product of the base cost and the historical congestion cost of a node u [7]. The number of signals using any given node of the RRG can vary vastly during the routing process; very often, no signal will use it. Hence, to make the A* heuristic *admissible*, the lookahead—based on the cost of the shortest possible path between all pairs of CLBs in the given FPGA architecture [1]—has to be computed while assuming that all nodes are unused. It has long been thought that increasing $pres\_fac$ at every iteration by multiplying it by some constant > 1 (1.3 being the default in VTR-8 [1]) leads to removing congestion faster by making the router focus on eliminating congested hotspots rather than trying to let each signal keep its preferred routing resources [5]. However, when the design is severely congested, it takes many iterations to remove overlaps, causing an exponential increase in $pres\_fac$ and making it overshadow the lookahead. As a result, the shortest path algorithm is reduced from A* to Dijkstra. Hence, on large designs, increasing $pres\_fac$ may help in reducing the number of iterations taken to converge, but the time taken to find each individual path is often increased.

To overcome the issue, we need to prevent $pres\_fac$ from increasing beyond a point that would make A* ineffective. We do so by fixing it to 5—the value previously used by Zha and Li [22]—and retaining it throughout the routing process. Contrary to us, Zha and Li fixed $pres\_fac$ to a single value to reduce critical path optimization variability and not improve runtime. In fact, they even mention that fixing $pres\_fac$ could often lead to an increase in runtime [22], which is true for small designs with low congestion that have short paths and take only a relatively small number of iterations to converge—these are precisely the circuits for which PathFinder was originally developed [5].

Our second modification concerns the strategy for ripping up and rerouting congested signals. The latest VTR release incorporates an efficient routing strategy known as the *Adaptive Incremental Router*, which reduces computational effort during each routing iteration by selectively ripping up sub-trees of nets and subsequently rerouting them [23]. By default, VPR applies partial rip-up only to nets with a fanout exceeding the threshold of 16 [1]. However, we have observed that reducing the threshold reduces router runtime. Consequently, we set the $min\_incremental\_reroute\_fanout$ threshold to the minimum possible value of one. After incorporating the above-described enhancements, we observe, on average, a reduction of heap pushes and pops by 2.3× and 3.6×, respectively, which leads to a 4.9× reduced runtime. In the remainder of the paper, we use the modified parameters and report all the results assuming this new, enhanced baseline.

V. Quantifying the IIB Routing Bottleneck

Having tuned VPR to route highly-congested benchmark circuits efficiently, we are ready to measure the impact of a realistic sparse IIB architecture on router performance. Starting from the architecture model described in Section II-A, we create two target RRGs. In the first, IIB connectivity is modeled according to Fig. 2 and referred to as *IIB-6-2L* (six-input multiplexers, two levels). In the second,

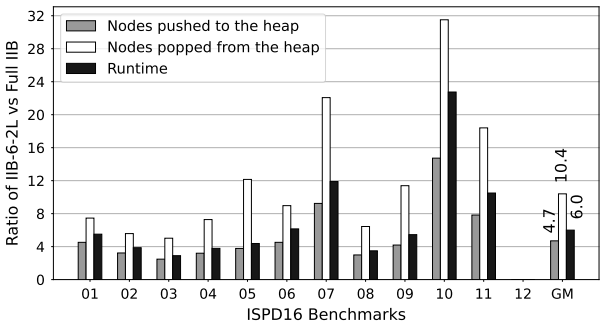


Fig. 4: The ratio of the number of heap pushes, pops, and routing runtime for IIB-6-2L vs. Full IIB across ISPD16 benchmarks.

we effectively create a full-crossbar IIB by connecting all adjacent length-1 and length-2 routing channel wires to each target pin inside the CLB (i.e., each LUT input and each FF direct-access pin). We compare the runtime metrics of *IIB-6-2L* to this full-crossbar IIB (*Full IIB*) to estimate the routing effort required to legalize through IIB-6-2L. In fact, the comparison is even slightly pessimistic since the targets are CLB and not IIB inputs, meaning that every shortest path receives an extra level of node expansions before it is found.

Fig. 4 summarizes the results (the data for benchmark 12 is not shown because, after 1,000 iterations, the routing did not converge to a legal solution). When the router is instructed to complete the entire routing, including the part through the IIB (i.e., on the IIB-6-2L RRG), the geometric mean (GM) of heap pushes, pops, and runtime increases by 4.7×, 10.4×, and 6×, respectively, compared to the situation when we stop the routing at the IIB boundary (i.e., on the Full IIB RRG). This confirms our intuition that routing through the IIB is computationally expensive, contributing to a large runtime increase. In the next section, we introduce our multi-stage routing approach for resolving this bottleneck and speeding up the routing.

VI. Multi-Stage Routing

A. Overall Idea

To leverage the runtime advantage associated with bypassing IIBs during routing, we divide the routing into three steps, illustrated in Fig. 5. In *Step 1*, signals are routed through the general interconnect, stopping at IIB boundaries (example signals are shown in blue at the top right in Fig. 5). At the end of this step, no congestion is present in the general interconnect, and IIB input pins for intra-CLB routing are determined.

In the following *Step 2*, routing inside IIBs is performed. It is important to note here that, because routing problems associated with IIBs are entirely independent, they can be processed in parallel and in any order. This is unlike other partitioning-based parallel routing strategies [3], where perfect independence between routing sub-problems is never guaranteed. Here, the runtime efficiency of intra-CLB routing in Step 2 is largely determined by the number of available processing cores.

Because IIB inputs are fixed in Step 1 and IIB connectivity is sparse, it is conceivable that, for some IIBs, Step 2 returns only a partially legal solution, that is, with nets overlapping inside the IIB. To account for this issue, at the end of Step 2, we combine routing solutions from Step 1 and Step 2. In the last step, *Step-3*, overlapping nets (in red in Fig. 5) are ripped up and incrementally rerouted all the way to LUT and FF inputs. The incremental rerouting can also lead to ripping up and rerouting of legal nets.

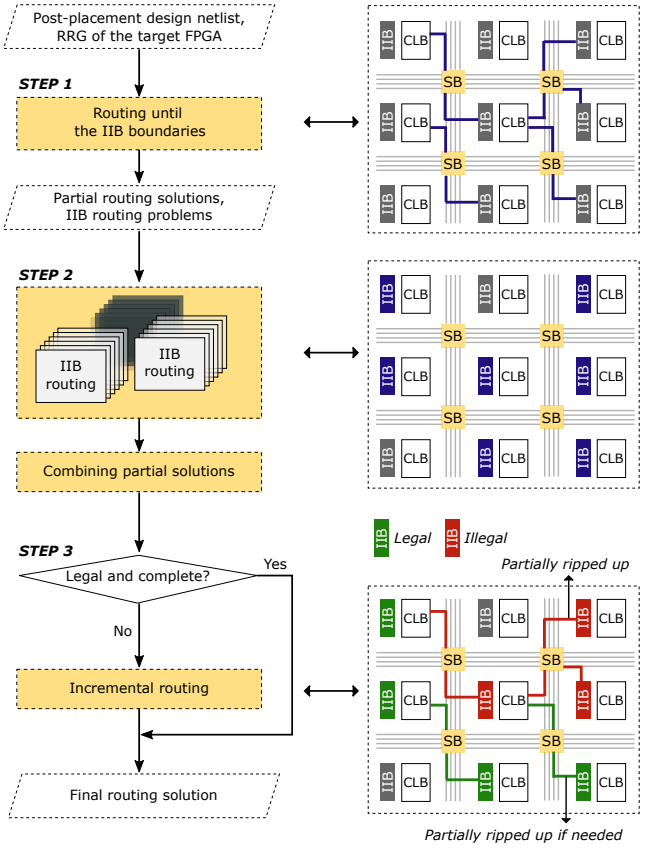


Fig. 5: Proposed multi-stage routing flow.

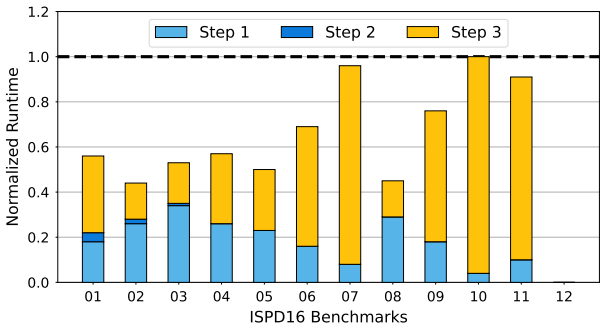
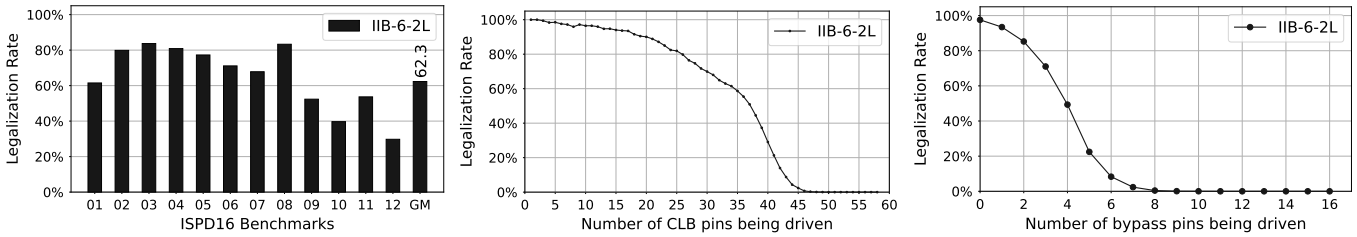


Fig. 6: Runtime comparison for IIB-6-2L: Multi-Stage Router vs. Single-Stage Router. Geomean speedup is 1.55×.

B. Implementation and Results

For implementation purposes, in Step 1 of our Multi-Stage Router, we use *Full IIB*, which gives a fair estimate of routing effort until the IIB boundaries, as described in Section V. The input IIB pins determined at the end of Step 1 serve as starting points for intra-CLB routing in Step 2. To overcome the difficulty of adapting VPR for routing only inside IIBs, we wrote a multi-threaded implementation of the PathFinder algorithm in C++. With the routing resource graph describing only the IIB topology and the maximum number of PathFinder iterations set to 50, we perform IIB routing in Step 2; we set the number of threads to 48, the highest supported by our experimental setup (see Section III). Once the partially completed routing solutions from Step 1 and Step 2 are obtained, we combine them to incrementally route in Step 3. At the start of Step 3, we load



(a) Legalization rate for ISPD16. (b) Legalization rate vs. driven CLB pins. (c) Legalization rate vs. driven bypass pins.

Fig. 7: Analysis of the IIB legalization rate in Step 2 of the Multi-Stage Router, for IIB-6-2L.

historical cost of the general interconnect from the last iteration of Step 1 and historical cost of all IIBs routed successfully in Step 2. In both Step 1 and Step 3, we run the enhanced VPR router with a limit of 1,000 iterations. While routing, we do not exclude clock and control signals.

Using the experimental setup presented in Section III, we routed ISPD16 benchmarks and measured the runtime for (a) baseline enhanced VTR router and (b) Multi-Stage Router. In the baseline router setup, the RRG includes a model of a realistic IIB (as described in Section V). In the remainder of the paper, we will refer to the described baseline router as *Single-Stage Router*.

The per-benchmark results for IIB-6-2L are visualized in Fig. 6. The overall height of each bar corresponds to the runtime of the Multi-Stage Router normalized to the corresponding Single-Stage Router runtime, while the dotted line corresponds to no runtime savings (i.e., normalized runtime equal to one). Hence, the bigger the gap between a bar and the dotted line, the better the speedup. Analyzing the obtained results, we first find that the geomean speedup is 1.55×, which is considerably lower than the 6× we reported in Section V. To understand why, let us return to Fig. 6, where each bar is composed of three parts: the ratio of the total runtime spent in Step 1 (light blue), Step 2 (dark blue), or Step 3 (yellow). Judging by the proportion of runtime spent in Step 3, not only that some IIBs were not legalized in Step 2 (which was anticipated), but the effort to incrementally route the combined solutions from Step 1 and Step 2 is far from negligible. Another interesting observation we can draw from Fig. 6 is that multi-threaded parallel IIB routing in Step 2 is completed in a comparably much shorter time (in 9.5 seconds, on average) than Step 1 or Step 3; this result confirms our reasoning in Section I-C that the absence of A* inside IIB is not a major concern when perceived in isolation. In the next section, we analyze the reasons behind the high runtime in Step 3 in further detail and propose ways to reduce it.

VII. Performance Analysis and Enhancements

A. What Limits the Exploitable Speedup?

If the assumption that intra-CLB routing can be performed independently from routing in the general interconnect always held, all IIBs would be successfully legalized during Step 2. However, as anticipated in Section I, due to the sparsity of the IIB architecture, congestion in some IIBs will not be resolvable without altering the routing through the general interconnect that was previously fixed during Step 1. The actual legalization rates which we measured on the ISPD16 benchmarks are plotted in Fig. 7a. At least those nets which still have overlaps in some IIBs have to be rerouted in Step 3, so that the final routing produced by the proposed algorithm is legal.

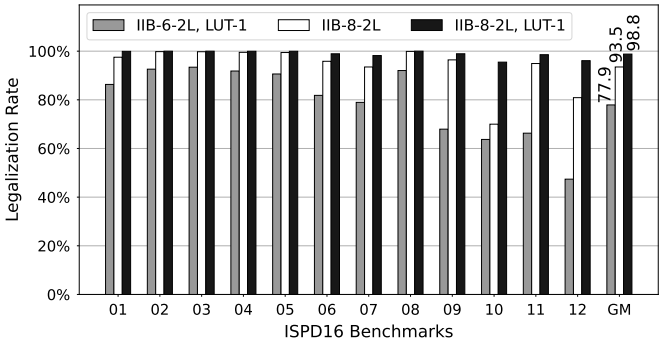


Fig. 8: Legalization rate comparison.

As shown in Fig. 6, low legalization rates lead to Step 3 taking a significant fraction of the total runtime. It is hence imperative to increase the legalization rate, ideally bringing it up to (very close to) 100%. To achieve this, we first need to understand what produces difficult intra-CLB problems. In Fig. 7b, we plot the legalization rate as a function of the total number of target pins in the CLB that are used (i.e., all used LUT inputs and all used direct-FF-access pins, labeled as *I* and *X* in Fig. 3). We can see that the legalization rate starts dropping significantly when more than 25 out of 64 target pins in the CLB are used. At first thought, it may seem that already very low pin utilization leads to very difficult problems. However, not all target pins cause the same amount of difficulties. Namely, LUT inputs are permutable, whereas if a signal has to reach a particular *I* or *X* input, this input cannot be exchanged for an alternative. Given that these direct-FF-access pins are driven by the bypass multiplexers, which are the only ones whose output is returned to the first level of the IIB (see Fig. 2), using too many of them, in turn, severely constrains the LUT-input routing problem as well. We can clearly see that from Fig. 7c, which plots the legalization rate as a function of the number of bypass pins that are used in the CLB: once half of these pins are used, the legalization rate essentially drops to zero.

B. Route-Through LUTs

The first method of reducing the pressure on bypass multiplexers we employ is purely algorithmic: whenever an FF is driven by a direct-access pin, but can instead be driven through an unused LUT, we resort to the second option. This way, the number of alternatives the router has to route to the FF increases thanks to the permutability of the LUT's inputs. We note that LUT *route-throughs*, which configure an unused LUT into a routing multiplexer where appropriate, are already used by Quartus [24]. Since we target applications that are not timing critical but where runtime—and hence

the high legalization rate in Step 2—is of great concern, we apply a route-through wherever intra-CLB placement creates an opportunity for it. After employing this optimization, which we call *LUT-1*, we observed a notable improvement in the legalization rate: the geometric mean increased from 62.3% (Fig. 7a) to 77.9% (Fig. 8). As a result, the geomean of the router runtime improved by 1.52×, leading to a total improvement of 2.36× compared to the single-stage router.

C. Enhanced IIB Architecture

The second method that we use to reduce the pressure on bypass multiplexers—but also the entire IIB structure in general—relies on a very simple architectural enhancement: we increase the size of all IIB multiplexers from 6:1 to 8:1. The two added multiplexer inputs effectively increase the number of available paths for routing the nets and help in alleviating congestion. Consequently, during each rip-up and reroute, there is an increase in the number of heap pushes. However, the total number of rip-ups required to resolve congestion decreases significantly, resulting in an overall reduction in total heap pushes and runtime.

To estimate the impact of this IIB modification on silicon area, we measure the increase in *Programmable Interconnect Points* (PIPs) [25] required to implement the enhanced architectures. A PIP structure fundamentally consists of a transmission gate controlled by a configuration memory cell. Adding two PIPs to each multiplexer that previously had six, without altering the multiplexer count, leads to a 33% increase in IIB area (measured approximately by the total number of PIPs). This increase may seem very significant; however, it should be noted that the IIB consumes about 30% of the logic tile area of a typical FPGA [12], while logic tiles, in turn, consume about 30% of the entire die area [26]. This means that the proposed increase amounts to merely 3% of the total die area, which is an estimate based on PIP counting that could likely be substantially reduced through the use of two-level multiplexers [26] and other layout optimizations.

Of course, this increase also inevitably leads to delay deterioration due to wire elongation and increased capacitive load. However, since we target applications that are not timing-critical, we do not consider this to be of major importance. In the case of other applications, the delay penalty could be mitigated through further layout optimizations and by adopting a more efficient architecture that specifically targets increasing the flexibility of bypass pins. That, however, goes beyond the scope of the present work.

With IIB-8-2L and LUT-1, the IIB legalization rate reaches 98.8% (Fig. 8). If we compare the runtime of the Multi-Stage Router with IIB-8-2L and LUT-1 with the runtime of the Multi-Stage router with IIB-6-2L and no enhancements, we obtain the results shown in Fig 9. The proportion of runtime spent in Step 3 is clearly reduced, which improves the geomean speedup further by 3.1×.

Finally, Fig. 10 compares the normalized runtime of the Multi-Stage Router with IIB-8-2L and LUT-1 to that of the Single-Stage Router with IIB-6-2L. Compared to Fig. 6, the share of the runtime of Step 3 is greatly reduced. As a result, our proposed router achieves 4.94× faster routing.

D. Detailed Results

Table II summarizes the speedup and normalized WL for the following four variants of experiments with the Multi-Stage Router: IIB-6-2L, IIB-6-2L with LUT-1, IIB-8-2L, and IIB-8-2L with LUT-1, compared to the absolute runtime and wirelength (WL) obtained with Single-Stage Router with IIB-6-2L.

Before discussing the results in Table II, let us briefly comment on ISPD benchmark 12. This design occupies 67,115 out of 67,200

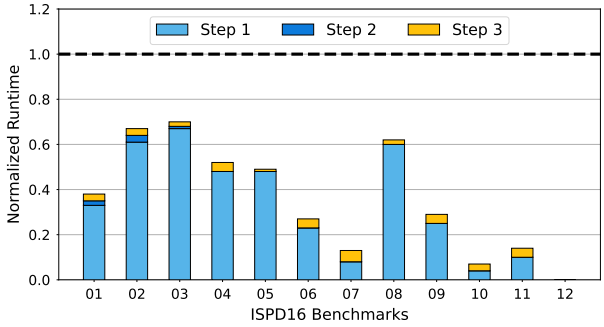


Fig. 9: Runtime comparison: Multi-Stage with IIB-8-2L and LUT-1 vs. Multi-Stage with IIB-6-2L. Geomean speedup is 3.1×.

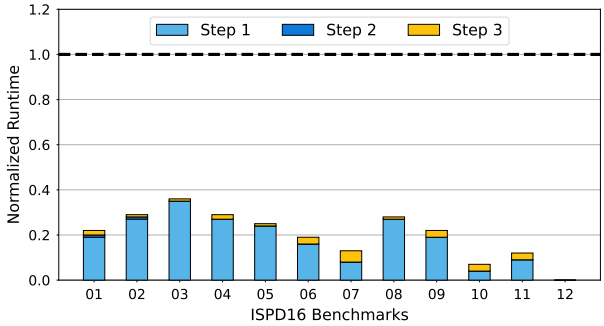


Fig. 10: Runtime comparison: Multi-Stage with IIB-8-2L and LUT-1 vs. Single-Stage with IIB-6-2L. Geomean speedup is 4.94×.

available CLBs, and it is the benchmark with the highest utilization of FFs (602K FFs versus ∼260K FFs, on average, by the remaining 11 benchmarks). In addition, the Rent exponent of 0.6 makes it undoubtedly one of the most difficult to route. In our experiments with IIB-6-2L (both with Single- and Multi-Stage Router), after ∼400 routing iterations, the number of congested nodes reduces to only a few (1–3). Incidentally, they are located at the edge of the FPGA, where the routability is inherently limited. As the congestion on these nodes is not eliminated in the subsequent routing iterations, we do not report the runtime or the WL for these experimental runs. However, once the IIB-6-2L is replaced with IIB-8-2L, the circuit successfully routes. The runtime (in minutes) and the total wirelength are reported in Table III, for reference. The example of benchmark 12 further confirms that the added cost of implementing IIB-8-2L quickly pays off: IIB-8-2L not only helps faster routing overall but also helps alleviate the congestion in the general interconnect.

Returning to data in Table II, we observe that the WL improves with every new and enhanced variant of the router. This is due to two factors: first, the routing in Step 1 of the Multi-Stage Router is not burdened by the intra-CLB congestion, which causes upstream rip-ups and detours in the general interconnect; second, less work is performed in Step 3, resulting in WL closer to the one obtained after Step 1.

VIII. Related Work

Moctar et al. also observed that routing inside sparse IIBs takes a significant amount of time and proposed to perform IIB routing separately [16]. Contrary to our approach, which is to route the IIBs once a legal routing in the general interconnect has been completed, Moctar et al. suggested routing each IIB before the general routing even starts, thus fixing the IIB entry pins for the signals in the

TABLE II: Speedup and wirelength (WL) for different configurations of our Multi-Stage Router with respect to the baseline Single-Stage Router (Enhanced VTR, IIB-6-2L).

Benchmark	Single-Stage Router		Multi-Stage Router							
	IIB-6-2L		IIB-6-2L		IIB-6-2L, LUT-1		IIB-8-2L		IIB-8-2L, LUT-1	
	Runtime (minutes)	WL	Speedup	WL (norm.)	Speedup	WL (norm.)	Speedup	WL (norm.)	Speedup	WL (norm.)
01	2.29	601,476	1.78	0.91	3.12	0.84	4.29	0.81	**4.64**	**0.81**
02	2.85	1,017,353	2.26	0.91	2.97	0.89	3.50	0.88	**3.42**	**0.88**
03	12.82	4,757,537	1.89	0.95	2.30	0.94	2.78	0.93	**2.71**	**0.93**
04	28.09	7,772,544	1.75	0.97	2.18	0.96	3.43	0.96	**3.36**	**0.96**
05	142.55	13,006,369	1.98	0.99	2.81	0.98	4.11	0.98	**4.05**	**0.98**
06	50.74	8,817,464	1.44	0.97	2.28	0.94	4.33	0.91	**5.22**	**0.91**
07	384.13	14,034,613	1.04	0.98	1.59	0.96	4.95	0.94	**7.50**	**0.94**
08	35.46	11,433,301	2.20	0.96	2.69	0.96	3.27	0.95	**3.56**	**0.95**
09	278.95	16,930,863	1.31	0.99	1.76	0.97	4.25	0.95	**4.61**	**0.94**
10	339.90	11,935,708	1.00	0.99	3.18	0.88	5.91	0.84	**13.68**	**0.82**
11	387.78	14,617,538	1.11	0.98	1.77	0.96	5.81	0.92	**8.27**	**0.92**
Geomean	52.60	6,704,329	1.55	0.96	2.36	0.93	4.13	0.91	**4.94**	**0.91**

TABLE III: Routing runtime and wirelength (WL) for benchmark 12.

Benchmark	IIB-8-2L		IIB-8-2L, LUT-1	
	Runtime (minutes)	WL	Runtime (minutes)	WL
12	42.60	8,696,445	25.04	8,546,902

second step [16]. This approach is very effective at reducing runtime when architectures have generous connectivity compared to the requirements of the circuits, which was the case with those used by Moctar et al [16]. For historical reasons of the inability of VPR to route beyond IIB pins, as this was a trivial and thus neglected problem when IIBs were fully connected, around the same time as Moctar et al., Luu et al. developed an almost identical approach and integrated it in VTR-7 [15].

Unfortunately, fixing all IIB pins a priori, without any information about the congestion in the general interconnect, creates very difficult inter-cluster routing problems that drastically increase runtime and sometimes even make it impossible to resolve congestion in realistic architectures. To confirm the above reasoning with VPR, we added a fully connected IIB and connected each input of the IIB to four channel wires. Once instructed to route, VPR first fixes the choice of the pins at the IIB boundaries (equivalent to routing inside IIBs first, but at no runtime cost), and only then routes in the general interconnect. As expected, the router failed to resolve congestion for some benchmarks (7, 9, and 12). Taking the runtime obtained for the remaining benchmarks and comparing it to the runtime of the Single-Stage router with IIB-6-2L (second column in Table II), we observed that fixing IIB pins a priori results in 1.8× longer routing, on average.

Recently, Wang et al. also identified intra-cluster routing as a major bottleneck, which they tried to resolve by splitting the routing process over multiple, albeit less clearly separated stages [27]. Although the speedups that the authors report are more significant than what we measured in this work, a note should be taken that the baseline with which they make their comparison is vastly inferior to the one that we use. With their baseline setup, four ISPD16 benchmark circuits are not routable in 24h, while the remaining seven take a total of 2168 minutes to route. As a comparison, in our baseline, routing these circuits takes 472 minutes in total. We believe that it is much easier to improve a baseline of so significantly lower quality and that the baseline is, in fact, the main cause for a higher *relative* runtime reduction than ours.

Considering the final runtime numbers, our Multi-Stage Router with IIB-8-2L and LUT-1 is 4.3× faster in completing the routing of all ISPD16 circuits than that proposed by Wang et al. With IIB-6-2L and LUT-1, the runtimes of our Multi-Stage Router are comparable to those reported by Wang et al. Although, due to likely different FPGA architectures (the IIB topology not being described in detail in [27]) and the fact that, unlike us, Wang et al. do not route clock and control signals (which contribute to the congestion in the general interconnect), it is not possible to compare these numbers directly. We believe the similarity of the results of Wang et al. and ours (with IIB-6-2L and LUT-1) clearly demonstrates that many of the complex algorithmic techniques proposed by Wang et al. are not really needed to resolve the runtime bottleneck. This should not come as too big of a surprise, though. In Section V, we observed that the most significant runtime savings could be obtained from avoiding the need to resolve congestion in target IIBs each time a signal changes its path through the general interconnect; once this large source of required work is removed, additional improvements of the general interconnect routing runtime—for instance using the global/detailed routing split that Wang et al. proposed—become less important. Nevertheless, we note that in our partitioning approach, any such technique is entirely orthogonal and can be used to achieve further speedups.

IX. Conclusions

We identified a routing runtime bottleneck within the CLB when routing through sparse input multiplexing structures (called the IIB) representative of the current commercial state-of-the-art FPGAs. We propose a Multi-Stage Router where we decouple the routing to the inputs of the IIB from the routing inside the IIBs (to reach actual CLB pins). The decoupling leads to independent routing problems at the CLB level, unlocking significant parallelism. A router runtime speedup of 2.36× is achieved over an enhanced state-of-the-art Single-Stage Router. We propose enhancements to the IIB architecture in light of the Multi-Stage Router algorithm to improve the IIB legalization rates, achieving speedups of 4.94× without loss of quality. We believe the ideas and results presented here open up new avenues for architecture-aware design of runtime-efficient routing algorithms for FPGAs.

REFERENCES

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

[2] E. Vansteenkiste, A. Kaviani, and H. Fraisse, "Analyzing the divide between FPGA academic and commercial results," in *Proc. of the 2015 International Conference on Field Programmable Technology*, Queenstown, New Zealand, Dec. 2015, pp. 96–103.

[3] M. Stojilović, "Parallel FPGA routing: Survey and challenges," in *Proc. of the 27th International Conference on Field Programmable Logic and Applications*, Ghent, Belgium, Sep. 2017, pp. 1–8.

[4] M. An, J. G. Steffan, and V. Betz, "Speeding up FPGA placement: Parallel algorithms and methods," in *Proc. of 2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines*, Boston, MA, USA, May 2014, pp. 178–85.

[5] L. McMurchie and C. Ebeling, "PathFinder: A negotiation-based performance-driven router for FPGAs," in *Proc. of the 3th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, Monterey, CA, Feb. 1995, pp. 111–17.

[6] S. Kaptanoglu, "Pathfinder: A negotiation-based performance-driven router for FPGAs," FPGA and Reconfigurable Computing Hall-of-Fame Endorsement. Available: http://tcfpga.org/fpga20/p32.pdf, 2012.

[7] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*. Norwell, MA, USA: Kluwer Academic publishers, 1999.

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

[9] A. Boutros and V. Betz, "FPGA architecture: Principles and progression," *IEEE Circuits and Systems Magazine*, vol. 21, no. 2, pp. 4–29, 2021.

[10] S. Shrivastava, S. Nikolić, C. Ravishankar, D. Gaitonde, and M. Stojilović, "IIBLAST: Speeding up commercial FPGA routing by decoupling and mitigating the intra-CLB bottleneck—Artifacts," Available: https://doi.org/10.5281/zenodo.8267376, 2023.

[11] "UltraScale architecture configurable logic block," https://docs.xilinx.com/v/u/en-US/ug574-ultrascale-clb, AMD, accessed: 2023-5-19.

[12] D. Lewis, D. Cashman, M. Chan, J. Chromczak, G. Lai, A. Lee, T. Vanderhoek, and H. Yu, "Architectural enhancements in Stratix V™," in *Proc. of the 21st ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, Monterey, California, USA, Feb. 2013, pp. 147–56.

[13] G. Lemieux and D. A. Lewis, *Design of Interconnection Networks for Programmable Logic*. New York, NY: Springer, 2004.

[14] L. McMurchie and C. Ebeling, "Chapter 17 - PathFinder: A negotiation-based performance-driven router for FPGAs," in *Reconfigurable Computing*, ser. Systems on Silicon, S. Hauck and A. Dehon, Eds. Burlington, MA, USA: Morgan Kaufmann, 2008, pp. 365–81.

[15] J. Luu, "Architecture-aware packing and CAD infrastructure for field-programmable gate arrays," PhD Thesis, University of Toronto, 2014.

[16] Y. O. M. Moctar, G. G. F. Lemieux, and P. Brisk, "Routing algorithms for FPGAs with sparse intra-cluster routing crossbars," in *Proc. of the 22nd International Conference on Field Programmable Logic and Applications*, Oslo, Norvey, Aug. 2012, pp. 91–8.

[17] K. E. Murray, M. A. Elgammal, V. Betz, T. Ansell, K. Rothman, and A. Comodi, "SymbiFlow and VPR: An open-source design flow for commercial and novel FPGAs," *IEEE Micro*, vol. 40, no. 4, pp. 49–57, Jul. 2020.

[18] International Symposium on Physical Design (ISPD), "Routability-driven FPGA placement contest," https://www.ispd.cc/contests/16/ispd2016_contest.html, 2016, retrieved Aug. 2022.

[19] W. Li, S. Dhar, and D. Z. Pan, "Placements for ISPD16 benchmarks," http://wuxili.net/project/utplacef/, 2016, retrieved Sep. 2022.

[20] "Hardware-assisted verification market - by platform (hardware emulation, FPGA prototyping), by application (automotive, consumer electronics, industrial, aerospace & defense, medical, telecom) & global forecast, 2023-2032," https://www.gminsights.com/industry-analysis/hardware-assisted-verification-market, Global Market Insights, accessed: 2023-5-20.

[21] W. Li, S. Dhar, and D. Z. Pan, "UTPlaceF: A routability-driven FPGA placer with physical and congestion aware packing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 4, pp. 869–82, Apr. 2018.

[22] Y. Zha and J. Li, "Revisiting pathfinder routing algorithm," in *Proc. of the 30th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, Seaside, CA, USA, Feb. 2022, pp. 24–34.

[23] K. E. Murray, S. Zhong, and V. Betz, "Air: A fast but lazy timing-driven FPGA router," in *Proc. of the 25th Asia and South Pacific Design Automation Conference*, Beijing, China, Jan. 2020, pp. 338–44.

[24] I. Corporation, "Intel® Quartus® Prime Pro Edition Help version 21.4," 2023, retrieved May 2023.

[25] C. Lavin and A. Kaviani, "RapidWright: Enabling custom crafted implementations on FPGAs," in *Proc. of 2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines*, Boulder, CO, USA, May 2018, pp. 133–40.

[26] D. Lewis and J. Chromczak, "Process technology implications for FPGAs," in *2012 International Electron Devices Meeting*, San Francisco, CA, USA, Dec. 2012, pp. 25.2.1–25.2.4.

[27] J. Wang, J. Mai, Z. Di, and Y. Lin, "A robust FPGA router with concurrent intra-CLB rerouting," in *Proc. of the 28th Asia and South Pacific Design Automation Conference*, Tokyo, Japan, Jan. 2023, pp. 529–34.